8-8-2017

SiMAMT: A Framework for Strategy-Based Multi-Agent Multi-Team Systems

Dennis Michael Franklin

SiMAMT: A FRAMEWORK FOR STRATEGY-BASED MULTI-AGENT MULTI-TEAM SYSTEMS

by

D. MICHAEL FRANKLIN

Under the Direction of Xiaolin Hu, PhD

ABSTRACT

Multi-agent multi-team systems are commonly seen in environments where hierarchical layers of goals are at play. For example, theater-wide combat scenarios where multiple levels of command and control are required for proper execution of goals from the general to the foot soldier. Similar structures can be seen in game environments, where agents work together as teams to compete with other teams. The different agents within the same team must, while maintaining their own 'personality', work together and coordinate with each other to achieve a common team goal. This research develops strategy-based multi-agent multi-team systems, where strategy is framed as an instrument at the team level to coordinate the multiple agents of a team in a cohesive way. A formal specification of strategy and strategy-based multi-agent

multi-team systems is provided. A framework is developed called SiMAMT (strategy-based multi-agent multi-team systems). The different components of the framework, including strategy simulation, strategy inference, strategy evaluation, and strategy selection are described. A graph-matching approximation algorithm is also developed to support effective and efficient strategy inference. Examples and experimental results are given throughout to illustrate the proposed framework, including each of its composite elements, and its overall efficacy.

This research make several contributions to the field of multi-agent multi-team systems: a specification for strategy and strategy-based systems, and a framework for implementing them in real-world, interactive-time scenarios; a robust simulation space for such complex and intricate interaction; an approximation algorithm that allows for strategy inference within these systems in interactive-time; experimental results that verify the various sub-elements along with a full-scale integration experiment showing the efficacy of the proposed framework.

INDEX KEYWORDS: Artificial Intelligence, Multi-Agent Systems, Multi-Team Systems, Strategy, Machine Learning

SiMAMT: A Framework for Strategy-Based Multi-Agent Multi-Team Systems

by

D. Michael Franklin

A Dissertation Submitted in Partial Fulfillment of the Requirements for the

Doctor of Philosophy

Degree

College of Arts and Sciences

Georgia State University

2017

SiMAMT: A Framework for Strategy-Based Multi-Agent Multi-Team Systems

by

D. Michael Franklin

Committee Chair: Xiaolin Hu

Committee: Sushil Prasad
Xiaojun Cao
Mariana Montiel

Electronic Version Approved:

Office of Graduate Studies
College of Arts and Sciences
Georgia State University
August 2017

Dedication

To my wife, who was, and is, the optimal choice in the game of life and to my kids, who are the best robots I've ever built.

Acknowledgements

I would like to thank my wonderful and amazing family and the professors who guided me along the way. Each of you, in so many ways, have helped me more than words can express. Thanks for your leadership, guidance, and patience. I am honored to emulate you all as I help to train up the next generation of Computer Scientists and I will refer to you often.

Artificial Intelligence is neither artificial nor intelligent; rather, it is a simulation inspired by the natural and requires hard work, persistence, and design. - Me

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In multi-agent systems, where many agents are interacting within the same environment, there are many challenges presented in determining optimal action sets and group behaviors.

> "Learning to act in a multi-agent environment is a difficult problem since the normal definition of an optimal policy no longer applies. (Michael Bowling and Manuela Veloso, 2002)."

The rapid growth of the state-action sets required for each agent in a multi-agent system, and the aggregate effect of the combinatorial comparisons for each agent, create the need for a hierarchical approach to reasoning in such instances. By creating a hierarchy the level of computation required to approximate situational reasoning is greatly diminished. The levels of this structure are introduced here, with the caveat that these terms are variously defined depending on the field within which they are being used. For this research, the particular fields in consideration are Artificial Intelligence, Machine Learning, and Game Theory. First, some general

disambiguation is necessary concerning the term strategy before defining these terms within the context of this research.

The dictionary defines strategy as:

> **strategy:** *In (theoretical) circumstances of competition or conflict, as in the theory of games, decision theory, business administration, etc., a plan for successful action based on the rationality and interdependence of the moves of the opposing participants.* (Oxford University Press, a)

Or, in a more modern definition:

> **strategy:** *a plan of action designed to achieve a long-term or overall aim.* (Oxford University Press, b)

The word strategy comes from the Greek word $\sigma\tau\rho\alpha\tau\eta\gamma\iota\alpha$ (strategia), meaning generalship. While usually applied in the context of war, the word has meaning well beyond this area. This general definition guides the meaning as applied herein. This leads to the additional terms that, though common, will be defined to enhance the understanding of how the term strategy is used in this research and how it is differentiated from other similar terms.

> **state:** *a discrete or measurable value or set of values at a specific time* t *for a given environment.*

> **action:** *any movement from state to state or anything that would change a state.*

behavior: *a grouping of actions coordinated to achieve a desired goal, transition into a different set of states, or move through the state/action space.*

policy: *the mapping of an agent to a particular behavior.*

strategy: *the mapping of a team of agents to individual policies.*

1.1 Strategy as a Concept

How is strategy defined? This depends greatly on the context, and leads to ambiguity with regards to the term. In (Jorg Bewersdorff and Translated by David Kramer, 2005), strategy in the context of games is defined as "for a player, the complete set of instructions on how to play" (pg. 143). Further, in this same work, there is the notion of mutual equilibria among the various strategies implemented in multiplayer games (pg. 166-7). This indicates that the strategy is the ultimate guide on agent behavior, and that such strategies can conform to well known game theoretic principles. This means that the strategy inference principles presented herein have grounding in game theory and multi-agent systems. Strategies are often represented as a master plan, a plan that considers goals that may not be 'in play' currently, but that should be considered anyway. In this context, gaming, the terms strategy and policy are often intermingled. They are used as synonyms, but defined as separate terms, thus the ambiguity arises. The reality of how the term strategy is used in context, however, reveals the nuance that differentiates the two terms. When the term strategy is used

in determining the next move it is akin to and equivalent to the classical definition of a policy, whose aim is to select the correct next action from the set of actions given the current state. However, and informatively, when the term strategy is used to imply a longer view of the impact of making certain moves, or to examine and contemplate a series of moves, it is moving closer to the way the term strategy is classically defined. This is how the term is used in this research - how does an agent consider its current states and actions while keeping the longer term goals in view? Further, when there are game-changing moments that radically alter the decision process (i.e., entering a new phase of the game, changing the direction of play, etc.) there is a need for a policy change. The strategy-based system would use strategies to select the best policy from among the set of policies to ensure the optimal progression towards the endgame. This is formalized below.

In game theory, as mentioned above, strategy is often defined interchangeably with policy (Russell and Norvig, 2003). This research seeks to define strategy in a more classical sense. First, states and actions must be understood. Viewing the environment as the arena within which the agent is located, it is understood that the environment returns to the agent a state and some information about that state (perhaps a value for being in that state, or a reward for having reached that state). Given this state, the agent will then choose an action to take from this state that will lead to the next state (which may, cyclically, be the same state). Once this action has been taken, the environment returns to the agent the new state and the new value or reward for having entered that state. It should be noted that both the value

and the reward can be negative values, thus negating the need for distinction among the terms (e.g., reward vs. penalty). This interaction is shown in Figure 1.1 and simply provides a grounding for this simple idea of the exchange of an agent with its environment.

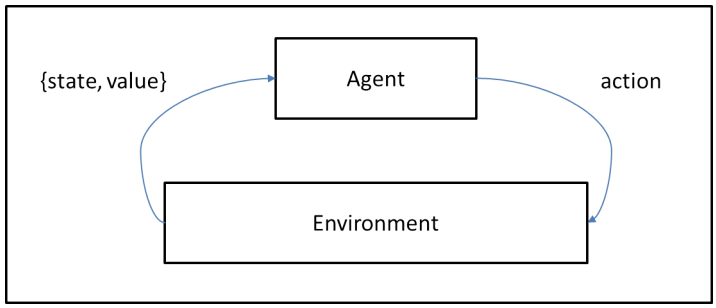


Figure 1.1: Agent / Environment Interaction

From this simplistic conception the agent has the means to explore its environment and learn which states are favorable, and with a learning mechanism in place, to discover which actions lead to the most favorable states. This simplistic model is learning a policy, a mapping of the best behavior for an agent (that, in turn, yields the best action, or chain of actions, to take from a certain state). Behaviors can also be viewed as a chain of actions that are all performed without interruption or further consideration. This allows reasoning at a higher level and reduces the state complexity. This leads to the way that strategies will be used to reduce the overall state space and considerations necessary for making decisions in larger scenarios, namely that the policies represent a chain of behaviors that in turn represent a chain of actions. Thus policy, in an environment that is static, will be used to govern the future choices of behaviors and actions for the agent from any given state.

This leads to understanding that this single-dimensional policy (i.e., in this state, always choose this action) is not expressive enough to allow for the complex sets of actions that are required for more complex environments (e.g., where the goal is moving, there is competition, or partial observability). In this scenario, it is recognizable that there is a need for multiple policies that are conditional (e.g., follow policy A as long as x is true, follow policy B otherwise). Understanding this condition necessitates an overarching policy, a super-policy, that allows for the governing of which policy to use under given conditions or when certain environmental variables are known. This is what is meant by a strategy, the selection of the best policy from among the set of policies given the current world state and goals. The strategy maps agents to policies, even when entire teams of agents are involved.

1.2 Strategy-Based Systems

With this introduction to strategy in mind, strategy-based system can use them as follows: given a current set of states and actions, and the current policy implemented, the strategy-based system will select from among the set of policies to engage the policy that best meets the long term goal (not just the short-term goal). That is, policies are designed to respond only to the current state and previous action (or previous few actions if memory is involved). Because of this, the policy tends to consider only the next best move with little consideration of the larger context (the preference for the short-term goal over the long-term goal). This is the second

differentiation between policies and strategies - strategies wish to consider the long-term goal and the transitions of policies to meet these goals. It should be noted that this may include sub-optimal action or policy selection as a necessary step to achieving the endgame (the long-term goal). Could policies simply be expanded to encompass these requirements by adding more global information, additional variables to consider, or a larger state set? The answer, of course, is in most cases, is yes; however, the cost of doing this is the loss of the ability to respond quickly to changes within the environment, the growing state space and corresponding computational complexity, and the vulnerability of such a policy to being outsmarted or tricked. Additionally, learning such policies would be exponentially more difficult (research to back this claim is part of this document).

For example, in the card game of Spades each team wishes to win as many 'tricks' or hands as possible. Each team bids on their expectation, but then tries to win at least that many or more. To punish teams that may underbid, the idea of 'sandbagging' is introduced. In this scenario, the cumulative effect of overbidding is penalized by the team losing points for 10 underbids. With this in place, a team may spend part of a round trying to win tricks, then shift to trying to lose tricks after their goal is reached. Here 'win all hands' and 'lose all hands' are policies (i.e., given a hand, draw the best card or draw the worst card). Determining which one is currently in force and when the transition occurs is strategic reasoning. As mentioned before, in this scenario an additional variable could be introduced that would perform this switch between policies. This variable would then be added to

the policy. However, this additional variable would make the policy more difficult to learn and the computation harder, especially as the number of agents increases and the scenario becomes more involved.

In another scenario, resource-bound auctions, the need for strategic reasoning becomes more apparent. Resource-bound auctions pit competing companies against each other bidding for finite resources. The challenge is that every bid won costs money to buy and even more to use. Likewise, every bid lost means that resources are given to a competitor. This means that the two goals, winning resources that can be funded and implemented and keeping the competitors from being able to fund their resources, are at odds with one another. One strategy-based approach is to buy early at whatever cost to get to market faster and ensure enough resources are available. Another strategy-based approach is to 'run up' the bids by bidding without the intention of purchasing. Perhaps another strategic option is to mix these two approaches. Each of these presents risk and rewards, and balancing them depends not only on individual goals and intentions but on understanding the goals and intentions of the other competitors. If the strategy-based approaches were enumerable, as suggested here in these few examples, and recognizable, one could use such information to great advantage. 'Running up the bid' is not a viable strategic approach if the competition knows that the company has no intention of buying. They will either outbid with confidence or force the purchase of unneeded expensive properties. This type of strategy, called bluffing in poker, can backfire if the bids are

not convincing or the competition knows that they have the winning hand. This is an important strength that strategic inference brings to such competitions.

1.3 Learning in Multi-Agent Multi-Team Environments

As an additional consideration it is imperative to consider the learning within such complex environments. Speaking in general terms, consider that each strategy is a mapping of each agent to their own policy, or from the known states to an optimal policy given certain environmental properties. This outside influence must also come from the environment, but it may not be related to being in a certain state (e.g., the light is on, the light is off, both state independent). This more closely resembles real-world scenarios as there are certain variables that are not related to the current state but are universally provided across the entire scope of the environment. One example may be to consider the game of musical chairs. While the music is playing (an environment variable that is not related to the current state of any of the players) the optimal policy may say to circle the chairs. When the music stops, the policy needs to shift to sitting in a chair as quickly as possible. It should be noted that these two policies are not just slightly different, or modifications of each other - they are opposites. How can an agent know to shift policies, or even that it needs to have multiple policies? While it is true that this global information, the state of the music (i.e., playing or not playing), could be included in one overarching policy, but would such a policy be easy to learn (based on time to learn or the number of samples that need to be seen before it can be learned)? Is there a way to accomplish

this learning without so much time elapsing or so many samples needed? Strategy-based systems exists for just this purpose. These two modalities (circling, sitting) result in strategies that are formed in similar ways to policies and can be thought of as an encapsulation of the divergence in a given policy. When a policy begins to 'break' under the pressure of new and different rewards being given, it should split into two separate policies rather than attempt to accommodate the massive change. This pressure can be thought of in terms of the momentum of the current policy and the counter force now acting to move it from this current path. When this pressure becomes too great (i.e., the variance of the new proposed change to the policy exceeds a threshold), a new policy is formed. This is then the lead intuition to understand both how various policies can be formed and how a strategy-based system can be taught to choose from among them.

Another rationale for the introduction of strategies is that the learning rate among the agents, especially in a multi-agent system, may differ (Chang Wook Kim and Seongwon Cho and Choong Woong Lee, 1995). This independence signifies the need for multiple independent policies to be in force (the results of such learning), and this governance again leads to the need for coordination among these policies; such governance is in the purview of strategy. This independence among individual agents does not carry to the multi-agent team – here there is a dependence. This further layer of complexity is the essence of multi-agent cooperation and similarly is challenging, if not impossible, to model with simple policies. Many papers offering insight into multi-agent learning require that this independence exist, as suggested

in (Chung and Lee, 1994). The authors suggest Competitive Learning (CL), Fuzzy Competitive Learning (FCL), Unsupervised FCL (UFCL) and an update to the Vector Quantization (VQ) introduced in (Chang Wook Kim and Seongwon Cho and Choong Woong Lee, 1995) called Fuzzy Learning VQ (FLVQ). While these ideas are out of scope for this research, they are shown here to demonstrate how modeling strategy, with its ability to maintain and administer different policies for each team and each agent, makes the practical implementation of such algorithms possible.

1.4 Strategy as a System

This leads to the hierarchical nature of strategy as indicated in (Headquarters, 2013). By way of example, consider the military concepts of a similar hierarchy. Speaking militarily, the lowest level of strategy is the action. This might involve any of the various single actions taking place in the theater of war, such as firing a rifle, fueling a plane, or replenishing supplies for troops. These actions are then grouped into tactics, like creating a forward push shaped like a spearhead into enemy forces, establishing an overwatch for a skirmish, or protecting valuable resources and their supply lines (Montgomery, 1968). Tactics involve the movement and usage of units within a particular theater and their correlated actions (University, 2012). Further, these tactics can be grouped into categories: general tactics (e.g., exploiting weather, fire attacks, reconnaissance), small unit tactics (e.g., shoot-and-scoot, infiltration, marching fire, ambush), offensive tactics (e.g., charge, skirmish, rapid dominance), defensive tactics (e.g., phalanx formation, counter battery fire,

fortification), deception (e.g., camouflage, stealth, feint), and many others. These tactics, then, can be grouped into strategies. A military strategy is 'the planning and execution of a contest between two adversaries (Publications, 2008).' NATO's definition of strategy is 'presenting the manner in which military power should be developed and applied to achieve national objectives or those of a group of nations (Office, 2015).' These same references state that these strategies (again, collections of tactics put into action to achieve a larger overall goal) are subjugated to policy. The policy is often the result of the political process within a government that empowers the military to enact these policies via their strategies (realized in tactics, resulting in actions). This hierarchy demonstrated the interrelation of the the various stages of strategic development and the interaction of the constituent elements. As indicated above, this scenario (military engagement) maps well to the subject being researched, namely multi-agent intelligence (policy) being realized through strategy and carried out via tactics (policies) that select the proper behaviors or sets of actions. While the nomenclature is similar, though not exact, the hierarchy is the same. This research proposes an Intelligence that selects which Strategy is in place while the Strategy selects which Policy is in place and the Policy selects the Behavior and that chooses the correct action to take from the given state. This matches the Political Policy to Strategy to Tactic decision making stratagem employed by the military. In particular, this research focuses on creating a hierarchical policy structure (intelligence to strategy to policy) rather than attempting to create one large, monolithic policy, in an attempt to encompass all of these various elements.

The strategies are thus subservient to an intelligence which considers both short-term and long-term goals, learning over time (multi-episodic learning), and experience in general. Intelligence would then balance each of these additional considerations to determine the best strategy to follow with respect to these larger goals, similarly to the way that a strategy-based system is selecting a particular strategy is selecting among the best policies locally.

By these definitions there is a hierarchy being formed where state-action pairs are being coalesced into behaviors, behaviors and single actions are being consolidated into policies, and these policies are being aggregated into strategies. Figure 1.2 shows this hierarchy and leads to intuitions regarding these elements.

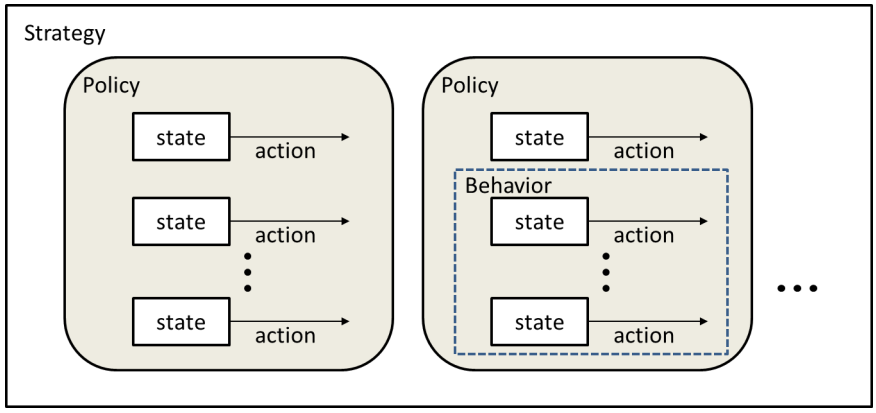


Figure 1.2: Strategy Hierarchy

Another advantage of implementing strategies is that the agent has the ability to plan moves farther in advance and with greater flexibility because the agent can plan moves based on the predicted moves of the other agents. This type of contingency planning offers advantages over Markov Decision Processes (MDP)(where only the current state and the next state are considered) as it can consider the past, the

present, and the most likely future states and values/rewards based on the most likely strategy in place. In a Markovian process, the window of time is limited by the order. In the usual case, only the current state and its observations are used. This window can be extended back a few moves, but it still is not a strategic view; rather, this backward view is more of a historical view, and this would be less useful without an overarching strategy in place to interpret this history (or to apply a semantic meaning to the previous steps beyond just the reward received for taking them). Further, this Markov process is not used to view the future. This type of inclusion, where the future values of states entered or actions taken is considered, can only be done in retrospect. In fact, it is the supposition of this research that considering intelligent future choices in the current state requires an understanding of the current policy and the strategy which governs it. *This ability, to offer a semantic meaning to past and future choices, is one of main contributions from the approach presented in this research.*

1.5 Multi-Agent Systems

As the complexity grows with the inclusion of multiple agents this problem becomes even more challenging. In the case of a multi-agent system the MDP is only partially observable, thus it becomes a Partially Observable Markov Decision Process (POMDP). The partial observability requires that beliefs are created (Simon Parson and Michael Woolridge, 2002), and this complex analysis quickly becomes intractable to compute (Simon Parson and Michael Woolridge, 2002). As a result, any interaction

among self-interested agents can be governed by game theory (Simon Parson and Michael Woolridge, 2002), and :

> "The classic game theoretic question asked of any particular multi-agent encounter is: What is the best - most rational - thing an agent can do? In most multi-agent encounters, the overall outcome will depend critically on the choices made by all agents in the scenario. This implies that in order for an agent to make the choice that optimises (sic) its outcome, it must reason *strategically*. That is, it must take into account the decisions that other agents may make, and must assume that they will act so as to optimise (sic) their own outcome. Game theory gives us a way of formalising (sic) and analyzing such concerns." (emphasis mine) (Simon Parson and Michael Woolridge, 2002)

Thus it can be seen that standard policy-only mechanization will not likely uncover an overarching policy capable of accounting for all of this complexity.

Once there is a strategy for an individual agent it then can be seen that it would be useful for the strategies of multiple agents to be coordinated into a master strategy for a team of agents. Further, it would be desirable for a team to coordinate its actions and to compare its team strategy with the strategies of other teams to determine if there is a better strategy they should implement to 'win the game'. *This type of analysis, determining a team's current strategy, comparing it to another team's strategy, then selecting a new strategy is the major focus of this research.*

By way of example, this research proposes to study a multi-team real-time strategic game, namely Speedball Paintball. There are professional leagues, collegiate leagues, and amateur leagues. Each event matches two 5-player teams in an enclosed field in an attempt to capture the other team's flag and return it to base. This task is made difficult by the team members firing paintballs at each other in an attempt to remove them from the game (a player is eliminated from the match when they are hit by a paintball). As a result, the team must make smart, real-time strategic choices about how they proceed, how they deploy, what role each team member takes, and many others. There are also options within these strategic choices that are on a sliding scale, like aggression or desperation. There are modalities such as being on offense or on defense. In short, there are many different elements that the team must both plan and execute. These choices are similar to the above-mentioned military strategy analysis. In fact, many of the most successful teams employ many of the exact same skirmish and close-quarter combat techniques, tactics, and strategies. This matching makes paintball an ideal testing format for the work in this research and it will be both described and tested in the experiments of this work. This explanation will describe the individual players and positions, tactics and techniques, strategies and variables, etc., that make up these matches.

There are many complications that arise in multi-agent systems. There is always the 'crowded workspace' issue that must be dealt with where each agent needs to avoid the others while still trying to accomplish its goals. Further, there are resource contention issues with space, time, rewards, and goals. This will apply if the agents

are competing or if they are exhibiting coordinated behavior. In this research, the primary concerns are two-fold: first, to exhibit a strategy that understands there are other agents exhibiting their own strategies; second, to either coordinate strategies with teammates or to compete against the strategies of opponents. It would also be important to determine if it is possible to discern if another agent is cooperating or competing without a priori knowledge. To accomplish this, there are several steps. It is critical to prove that strategic modeling is both possible and profitable. To do so two scenarios are examined: first, Roshambo and RPS-LS; second, a small-scale simulation. Each of these are explained in their experiments sections at the end of Chapter 3. Next, it is critical that simulations be constructed to show that the strategies are making the difference, and not simple probabilities. This applies to both inter-team coordination and intra-team cooperation and competition. Finally, it is essential that the multiple agents all factor in to the team strategy and the overall competition. The multi-agent system will be run by Overwatch (a contribution of this research, documented in Chapter 6, that allows simple robots to be used as intelligent agents in multi-agent multi-team experiments), so that will provide localization and navigation for each of the agents. Overwatch makes it possible to control each agent independently while enforcing policies, strategies, and overall gameplay.

By contrast, in many example machine learning problems it is typical for only one agent to be considered. The typical scenario is one agent interacting with a static environment and receiving consistent and deterministic rewards. This makes for a convenient academic case, but the reality is quite a bit more complex. One common

procedure is to treat the other agents in the system as if they are part of the system, but this can lead to a bad assumption (i.e., that other intelligent agents would behave as natural objects would) (Russell and Norvig, 2003). In multi-agent systems multiple entities are introduced who are also working within the same environment, each with its own agenda. For this research it is vital that these other agents be considered for what they really are – other 'intelligent' entities interacting with the environment in dynamic ways. What are their motivations or goals? Are they cooperating with the focus agent or competing against it? When strategy is considered, it becomes clear that the focus agent needs to understand exactly what these other agents are doing now, what they are likely to do next, and how to use that information to better its own condition. In the context of state-action pairs, this is a challenging problem. When only the current state and the next action of the other agents can be observed it is difficult to form a conclusive idea of how this information should be factored into the focus agent's decision. When this problem is framed within the context of strategies this becomes clearer. The focus agent can consider the most likely current strategy of the other agent by examining its most likely current policy, which was derived from its observed action set. With this information, it is possible for the focus agent to compare this probabilistic strategy with its own strategy and determine if they are compatible (i.e., working towards the same goal, matching action sets from given states, or not negating their own actions) or not. If they are, then this other agent can be considered to be cooperating. If the two strategies are opposing, then this other agent can be seen as competing. It is possible that there is no union in the

two strategies and, therefore, this additional agent is seen as neutral. Whichever case holds true, this gives the focus agent additional insight into its optimal choice, or at least its most probable choice if the inferred strategies are based on a maximum likelihood measure.

Agents cooperating within a multi-agent system are working to form a joint plan that coordinates their objectives by aligning their strategies and maximizing their performance. This may involve plan recognition, communication, or outside control. This inter-agent communication is critical for a multi-agent system to exhibit team behavior and for the possibility of emergent behavior (Russell and Norvig, 2003). Emergent behavior allows for synergistic opportunities that exceed what a single agent would be able to do in the same environment. This desirable trend towards higher functioning teams will be culminated in joint strategies and strategic cooperation.

The goal of this research is to have multi-agent systems that truly are considering the other agents within the system as intelligent agents who have their own strategies. This should provide a clearer set of goals for the focus agent and allow for better cooperation and competition.

1.6 Multi-Team Systems

Additionally, there are a number of factors related to being in a multi-team environment that further complicate the system. As with multi-agent teams, there is a difference between having many teams and having multi-team environments. In the first, the teams are just collections of individual agents working within the

same environment (complicated, but simplistic interactions between agents); in the latter, the teams themselves have cohesive goals and can work to cooperate with or compete against other such teams (both complicated and complex). This additional consideration means that as a team is selecting a strategy to implement they must now add an additional set of goals: choosing a strategy that helps the other teams in their alliance, choosing a strategy that impedes the other competing alliances, and choosing a strategy that maximizes their team efficacy. These various factors are implemented inside of the the SiMAMT framework in order to give the most accurate simulation of these complex systems possible. The goal of SiMAMT is to simulate, maintain, operate, and assess these multi-team interactions and to make real-time decisions that affect the teams.

One non-trivial portion of the development of SiMAMT was to be able to realize the simulation in the real-world. However, such interactions can be quite complex and difficult to manage (not to mention rather expensive to implement). This research worked to create an initial simulation to prove the concept of multi-agent interaction and then to realize that in robotic multi-agent teams. Of course, once the number of agents increased beyond the experimental range (in this case, beyond a hundred robots), the simulation had to take over and the development of SiMAMT was completed to handle such large-scale strategic interactions in real-time.

1.7 Game-Theoretic Implications

Game theory is often used to analyze the interactions of games that allow for simultaneous moves (rather than being turn-based) (Russell and Norvig, 2003). Game Theory is not limited to simple or trivial simulations, but 'is used in very serious decision making situations including bankruptcy proceedings, the auctioning of wireless frequency spectrums, product development and pricing decisions, and national defense, situations involving billions of dollars and hundreds of thousands of lives (Russell and Norvig, 2003)'. Even though the title includes the word 'game' this theory has vast and serious application. It is essential in environments where multiple agents are interacting with their own goals simultaneously. Further, much of the groundbreaking work in this field draws from the notable results in game theory (Bowling et al., 2004). The core of the work culminating in the Nash-equilibrium proved that games were an excellent method for understanding adversarial and cooperative environments with multiple players (each of whom have their own set of goals or objectives). Although a Markov Decision Process (MDP) may often be used in learning environments, they do not translate well to examples with multiple agents (Bowling et al., 2004). This complication arises because the next state of the system and the rewards given are determined by the joint action (Bowling et al., 2004). As a result, stochastic games are better suited for such environments.

This research aims to show that super-policies, or some hierarchical system of policies, is required to model this level of interaction; to that end, strategies that utilize game theoretic principles have been introduced.

1.8 Proposed Solution: SiMAMT

When multi-agent scenarios move beyond singular, short-term goals and into the realm of multi-layered strategies the complexity quickly scales out of the practical. Much of the research in multi-agent systems revolves around single-goal systems where multiple agents each work independently to achieve the same goal. This does not accurately model the real-world scenarios found in larger systems where each independent agent has their own initiatives but still works together to achieve team goals. These teams are also part of larger teams (e.g., units make up regiments, regiments make up battalions, etc.) that also have large-scale goals. As the hierarchy builds, the strategy becomes larger and extends further. We are proposing a framework to address this particular issue. The SiMAMT framework is designed to allow a hierarchical strategy structure that works at each level to enforce policies that work at that particular level. Each sub-level of the hierarchy then works at its' particular level order while considering the orders filtered down from the higher level. In this manner, the entire structure incorporates a multi-level strategy without having to use a large, monolithic policy (these large policies arise from applying small-scale solutions to much larger problems). When policies are allowed to grow in scale with the number of agents and the complexity of the system they become

computationally too complex to be applied, recognized, and changed in real-time. Strategy-based systems utilize group policies to aggregate the policies of individuals into a larger team policy (which we refer to as a strategy). Each of these strategies can be grouped together into a larger strategy at the next level. The SiMAMT framework creates a system to setup, model, control, and analyze multi-level strategies such as these.

SiMAMT seeks to model more complex multi-agent systems where there are multiple teams involved. Strategic interactions at this level are challenging and must be considered carefully. Considering the actions of other agents is foundational to strategy.

> "The classic game theoretic question asked of any particular multi-agent encounter is: What is the best - most rational - thing an agent can do? In most multi-agent encounters, the overall outcome will depend critically on the choices made by all agents in the scenario. This implies that in order for an agent to make the choice that optimises (sic) its outcome, it must reason strategically. That is, it must take into account the decisions that other agents may make, and must assume that they will act so as to optimise (sic) their own outcome. Game theory gives us a way of formalising (sic) and analyzing such concerns. (Simon Parson and Michael Woolridge, 2002)

> "...an agent...must (a) recognize that there are other agents, (b) compute some of the other agent's possible plans, (c) compute how the other agent's plans interact with its own plans, and (d) decide on the best action in view of these interactions. So [both competition and cooperation] require a model of the other agent's plans. (Russell and Norvig, 2003)

1.9 Research Goals

The goals for this research are as follows:

- Define and describe strategy-based systems
- Define and describe a framework for implementing strategy-based systems
- Conduct experiments of increasing scale to prove the performance of the new strategy-based systems

To achieve this, there are two major elements. First, we wish to introduce strategy as a concept and show how this concept can be built upon to create strategy-based systems. Second, we introduce a novel framework for implementing strategy-based systems, namely SiMAMT.

With regard to the first objective, we wish to show three major points with regards to strategy:

- Strategies offer significant performance enhancement to artificially intelligent agents, especially in highly complex environments such as multi-agent, multi-team environments

- Strategies can be recognized quickly, in nearly real-time (meaning within the given interval of the simulation)
- Agents utilizing strategy inference will outperform those agents that are not using strategy inference, even if those opponents were originally outperforming them

Classical machine learning requires repetitive trials and numerous iterations to begin to form a hypothesis as to the intended actions of a given agent. In this research, there are numerous methodologies employed in an attempt to reduce the number of examples needed to form a meaningful hypothesis. Chiefly, the system is considered hierarchically rather than monolithically. This greatly aids both the learning (it is now learning in a smaller environment with fewer features) and the implementation of the system. The traditional approach, learning one monolithic policy for the entire system, requires numerous iterations to learn. As the complexity of the system increases, the number of iterations required for learning tends to increase exponentially. The challenge arises from the difficulty created by the diversity of possible scenarios in which the machine learning algorithm is placed. Given enough time and stability, a machine learning algorithm can learn reasonably well in a fixed environment, but this does not replicate the real world very accurately. As a result, strategies offer an opportunity to encapsulate much of this policy-space in a compact representation. They have to be learned as well, but they are learning several smaller problems rather than one large one, and they are transmutable to another instance

of a similar problem. Additionally, they can be pre-built based on domain knowledge and then modified to suit the exact situation.

Next, we wish to introduce a framework to take advantage of these strategy concepts and to implement strategy-based systems. Once the concept of strategy has been proven, this research will then move on to creating strategy-based systems for simulations of multi-agent multi-team interaction. SiMAMT creates a realistic and complex environment in which the agents and teams of agents will act. The SiMAMT Framework is comprised of five distinct phases of processing. The first phase is the initialization phase called Strategic Modeling. SiMAMT is contingent on the ability to model strategy (i.e., to formulate complex systems of behavior into cohesive models). Once the models are in place the process commences with the Strategy Simulation module. The simulation module produces data that is fed into the Strategy Inference Engine (SIE) for processing. Once this data is consolidated and processed it is moved forward to the Evaluation Engine where it is analyzed. This evaluation is then forwarded to the Intelligent Strategy Selection Engine (ISSE) where a final decision is made as to the current strategy that should be in place given the evaluation. The cycle then repeats as the simulation continues until termination. This framework provides high-fidelity modeling of real-world interactions at each hierarchical level according to individual policies, behaviors, and group strategies.

The specification of the system continues in Chapter 3, but the final section of the Introduction offers background and developmental history of the transition from

strategy as a concept to strategy as a system for the interested reader. It is a multi-year story arc of the development of the concepts that have led to the the SiMAMT framework and it shows the work from its origins to its present day conceptualization. While it may be found to be interesting (and contains much corollary information and branching research topics), it is not required reading to understand the work presented in the rest of the research.

1.10 Background and Developmental History

The work contained in this research is the result of many years of investigation, experimentation, research, publications, and revision. As a consequence there is a lot of progression to show in the evolution of the final framework. In an effort not to confuse the reader, this background and history is offered here in a consolidated section for those who may be interested in the derivation of this system.

1.10.1 Formative Hypothesis: Game Theory in Multi-Agent Systems

There is much to be learned from the application of Game Theory and Decision Theory to Computer Science, and specifically to algorithms involved in multi-agent systems. The interaction, within the same environment, of many agents is complicated when they must each understand what the other agents are doing. As more agents are considered, and more detail about each agent is factored in, understanding their

interaction leads to larger and less manageable state spaces. This leads quickly to in-computability using normal algorithms and approaches where the system attempts to focus on all available information monolithically. One approach to resolving this large state-space search problem was proposed in (Soumya Paul and R. Ramanujam, 2010) whereby the researchers suggest a theoretical framework using imitation. In this work, they state that in large games there are optimizers (those agents who, due to superior resources, have a better understanding of the state of the game and thus perform optimally) that can be exploited by imitators (those agents with more limited resources that choose to imitate the optimizers rather than innovate). They incorporate principles derived from game theory to explain this interaction within the context of repeated normal form games. Ramanujam et al. propose using a finite graph with infinite moves, a small set of known strategies, and turn based progression (each player takes a turn rather than simultaneous moves being allowed). Starting from the foundation laid in (Soumya Paul and R. Ramanujam, 2010), this research contributes a framework that moves their work from the theoretical to the practical. Their proposed mini-game will be realized using small robots as the players that must follow the same rules. This will lead to a larger scale solution that emulates an increasingly complex environment within a simulation system. In the end, their proposal states that, given enough time, the imitators can improve their performance (though not to the level of the optimizers) using fewer resources. This should result in the robots playing the game with improved performance (higher overall reward). This initial insight led to the strategy-based systems implemented in

the proposed framework. While this reference work focuses on a small, single-agent team, theoretical environment, the proposed framework works in large, multi-agent, multi-team, practical environments.

Their simple mini-game, as dictated in (Soumya Paul and R. Ramanujam, 2010), will be expanded in three ways: first, allowing simultaneous moves; second, co-located players (more than one player in a given location as opposed to the exclusivity of each location in the original proposal); third, finite length for the game. This first change adds uncertainty due to simultaneity and requires probabilistic modeling. The second change adds complexity to the algorithmic requirements because the state space is not narrowed down by spaces being held by other players. The third change adjusts the time scale allotted for realizing what other players are doing, imitating them, and increasing performance. As Ramanujam et al. proposed their game it is unbounded in length. Unbounded length allows more time for the slow gains realized by the imitators to pay off. In a finite game (which could also represent a single episode of a longer, perhaps infinite game) the time for the payoff to factor in is greatly reduced. As a result, a different representation scheme is required for strategies.

To understand the evolution of the proposed strategy inference (utilized in this research and in the final experiments), it is necessary to bridge the gap from some theoretical background work (Soumya Paul and R. Ramanujam, 2010, 2011a,b), to an applied system that can be constructed, run, and evaluated. Many of the issues found with those early theoretical models are described here and the reason for abandoning

this approach and shifting to the final, novel, proposed approach is also elucidated here.

For single-agent systems, moving from the theoretical to the generic, each agent's strategy must be considered and compared. In this manner the focus agent's strategy, σ_1, can be compared to the most likely strategies of the other agents. The steps of this strategy inference are shown in Table 1.1.

Step	Techniques
Develop a strategy	Observation, Expert System, Coach
Choose optimal move independently	Pre-determined policy from strategy
Choose most likely moves of other agents	FSA and Belief Network
Match best possible move	Re-examine policy and select

Table 1.1: Steps and Techniques for Strategy Inference

1.10.2 Specification

In an effort to help the reader, the evolution of the concepts of strategies and policies is given next. This specification uses some similar terminology and symbology with the final product (the effort of this research), but it is not the same. The explanation follows within each section. In summary, the original concept, as communicated in the reference papers, was sufficient for a technical discussion, but not sufficient for a real-world system. The reference system has some issues that prevent it from being implemented in practice. However, to aid the research community, this concept is presented in its original form and using the reference notation. Once this is relayed,

explained, and discussed, the new concepts, formalism, and notation are introduced (introduced generally here, formally presented in Chapter 3).

A formalization of strategy, presented in (Soumya Paul and R. Ramanujam, 2010), consolidates this type of strategic interaction. This paper is written using the game-theoretic definition of strategy, so it is similar in practice to a policy, but the intention is larger than this standard definition. This research seeks to move their work from the theoretical and policy-bound to the practical and strategy-bound. In this case, the game is considered as a turn-based game of unbounded duration. Rather than give a simple reference, the paper's main points are included here in a paraphrase with some interpretation.

The game in (Soumya Paul and R. Ramanujam, 2010) is mapped as a finite graph. For any positive integer n, let $[n] = 1, .., n$.

Definition 1 *Let $n \in \mathbb{N}, n > 1$. An n-player game arena is a directed graph $\mathcal{G} = (V_1, \ldots, V_n, A, E)$, where V_i are finite sets of game positions with $V_i \cap V_j = \emptyset$ for $i \neq j, V = \cup_{i \in [n]} V_i$, A is a finite set of moves, and $E \subseteq (V \times A \times V)$ is the move relation that satisfies the following condition:*

1. *For every $v, v_1, v_2 \in V$ and $a, b \in A, if (v, a, v_1) \in E$ and $(v, b, v_2) \in E$ then $a \neq b$.*

2. *For every $v \in V$, there exists $a \in A$ and $v' \in V$ such that $(v, a, v') \in E$.*

When an initial position $v_0 \in V$ is specified, we call $(\mathcal{G}, v_0)$ an initialized arena or just an arena.

However, there are issues with this theoretical foundation when the game is moved to a real-world simulation. Namely, no models are offered for implementation, there are no probabilistic transitions, and only one player can be in any particular state at one time. Since this research seeks to investigate systems with probabilistic progression with strong modeling of multi-agent teams, such limitations require another formulation of the problem.

This research creates an arena defined as this graph $\mathcal{G}$ with the players (the agents) as the vertices and the possible moves as the edges. This definition allows the definition of the neighbors of the players to be those who are located along the connected edges of the graph. These neighbors, vE, are the singularly located so that there are no other players at this same vertex. Additionally, the edges denote those actions, or moves, which are possible for the player at this state and each is available to this player as long as it is enabled (i.e., vertex v' is unoccupied). These moves can then be chained together as a sequence of moves along this finite graph which will be called a play, labeled as $v_i \overset{a_i+1}{\rightarrow} v_{i+1}$ for $i \in \mathbb{N}$. Since there is no 'game board' in the scenario being used in this research this series of moves is a projection into a space of possible and inter-related moves. This complex scenario of moves is really just a mapping of enabled moves for a player that do not overlap their own teammates moves nor create a co-location with a player (i.e., choosing the identical unique action) from any team. It is important to note that there may be identical moves that are allowable, such as two players claiming rewards from a pool of rewards, and moves that are not allowable, such as two players trying to claim the same reward.

For example, for two players i and j, their possible moves are $v_i = \{1, 2, 3, ..., n\}$ and $v_j = \{1, 2, 3, ..., m\}$, respectively. The sets of moves represent the action taken by a player at each timestep $t \in T$, and there is always a 'no move' option. In this manner, there is a move made by each player at each time step, even if it is a 'no move'. Thus, the moves are aligned within the vector of moves such that $v_i[a_1]$ and $v_j[a_1]$ occur contemporaneously. The game $\mathcal{G}$ constrains the set of moves that enforces the rules of the game (i.e., the set of policies that govern the environment, which in itself can be thought of as another implicit player (Soumya Paul and R. Ramanujam, 2011a) whose strategy must be considered) and ensures that $v_i[a_n] \neq v_j[a_m], \forall t \in T$. Similarly, there are always $a \in A$ s.t. $\forall\ v \in V\ \exists\ a'$, so there are no absorbing states in the game from which a player cannot return. Each of these players is implementing a strategy, σ_1 and σ_2, respectively. These strategies are composed of policies, π_i and π_j, where $\pi_{1..n} \in \Pi_i$ and $\pi_{1..m} \in \Pi_j$. Therefore v_i and v_j are following policies parceled out to them by their strategies.

1.10.3 Methodology

The application of strategy can, and usually does, vary by domain. While this research seeks to unify a clearer definition of strategy as that overarching policy that determines which policy is in place, not all papers use the same concept. For example, in this particular paper cited, strategy is used in both senses. The following formulations were helpful in discovering the concepts of strategy and the implementation of strategy-based systems, but these equations are not those used by

the final research (again, these are found in Chapter 3). Though it is confusing, the symbols and terms are presented here as used in the reference work even though those symbols will be redefined and reused in the final research. However, the following is the definition given as cited, which will be notated and discussed afterwards. In this example game, a strategy, σ, is defined for player i in a partial function

$$\sigma_i : VA^* \rightarrow A \tag{1.1}$$

where V represents the set of all players, A the set of all actions (moves). The '*' indicates the Kleene star (or Kleene set or closure), or, mathematically, a free monoid. It is used in this example and in this research as a set of finite-length strings generated from the alphabet or from the set of strings. Here, it is the set of finite-length strings generated from A. This could represent a history of all moves or the total set of all possible move combinations. For example, if the actions available were up, down, left, or right, designated by $\{U, D, L, R\}$, then acceptable A^* would include $\{\varepsilon\}$, (an empty set, no actions), $\{U, U, U, R\}$, $\{D, L, R, D, U, \varepsilon, D, R\}$, etc. This equation, restated from the referenced material, can be expressed in the terms introduced in this research using the definitions stated earlier in this section. This better aligned version of the equation is shown below:

$$\sigma_i : v_j a^* \rightarrow a', \forall v \in V, \forall a \in A_{v_j} \tag{1.2}$$

Thus σ_i produces the next action for the given player. As noted above, using the term strategy here is replaceable with the term policy, as traditionally the policy gives the next action of the player. Additionally, σ is bounded memory (Wilson, Andrea, 2002) if there exists a finite state transducer (FST) $\mathcal{A}_s = (M, \delta, g, m_o)$ where M is the memory of the strategy and is a finite set of states, $m_0 \in M$ is the initial state of the memory, $\delta : A \times M \to M$ is the memory update function, and $g : V_i \times M \to A$ is the action taken by the player such that $\forall v \in V$, and $m \in M$, $g(v, m)$ is enabled at v and the following condition holds: given $v \in V_i$, when $u = a_1, ..., a_k \in A^*$ is a partial play from v, $\sigma(vu)$ is defined, $\sigma_i(vu) = g(v[u], m_k)$, where m_k is determined by: $m_{i+1} = \delta(a_{i+1}, m)$ for $0 \leq i < k$. A strategy can be Markovian if M is a singleton, or Markov Order 1,2,3 with M being {1,2,3} respectively. This memory quality is an expansion of 1.1.

Definition 2 *Given a strategy profile $\bar{\sigma} = (\sigma_1, ..., \sigma_n)$ for n players let $\rho_{\bar{\sigma}}$ denote the unique play in ($\mathcal{G}$,v_0) conforming to $\bar{\sigma}$. A profile $\bar{\sigma}$ is called a Nash equilibrium in ($\mathcal{G}$,$v_0, \prec_1, .., \prec_n$) if for every player i and for every other strategy σ_i' of player i, $inf(\rho_{(\bar{\sigma}_{-i}, \sigma_i')}) \preceq_i inf(\rho_{\bar{\sigma}})$.*

This research seeks to move this cloudy definition of strategy back to the formal presented in this research. In this context, the strategy suggests the profile which is in play currently. This strategy profile is synonymous with a policy that is selected by the strategy to be currently in force. As their game is theoretical it is not reasonable to derive the complete strategy and related policies, but the example is provided to show the general idea of the game and the type of reasoning that should be possible

given proper strategies. This research provides an enhanced and practical version of the game, modified to show the distinction of policies and strategies as applied to a real game. This game is presented in the experiments described herein.

Before it can be represented strategy must be defined. To understand strategy it is essential to define each of the relevant terms. As these terms have already been defined in a natural language discussion format it is time to formalize these definitions. First, we wish to establish the generic principles designed to bridge the gap from the theoretical to the practical. Second, once this general formulation is clear, we will move from this to the practical application. This second result will be the formulation that realizes the goal of moving the concept of strategy into the real world and handles the complexity of multi-agent and multi-team environments.

The paper cited, (Soumya Paul and R. Ramanujam, 2010), offers some insight into a general strategy model. This formulation will be expounded here and explained, but the specific proposed methodology and model of strategy used in this research is detailed in Chapter 3. This section seeks to move from the Related Works citations to the generic formulation. Since a strategy is comprised of various policies that have defined states and actions, these must be clarified. A state, s, is a discrete or measurable value or set of values at a specific time t. All s are in S, the set of all states. An action, a, is any transition from state s to a subsequent state s', or anything that changes a state. All a are in A, the set of all possible actions. A policy, π, is a mapping from $S \rightarrow A$, or that chooses a', the next action, from A optimally for all $s \in S$ (1.3, generically). There are several ways this can be derived and several

considerations for the method used. First, there is the generic formulation shown in Equation 1.4. To evaluate the results of taking each of the actions from a given state, and thus determine the optimal action, a valuation function is needed. This function (e.g., 1.5) generally determines a value of taking a certain action from a certain state by capturing the reward of that action. It may additionally consider the current action plus the previous action or actions. While the valuation function, V, is left generalized here, it is defined specifically during implementation, and will be shown in particular for each of the following examples in this section. This function is critical to the success of the policy and essential for it to have decidability from among the various available actions. Additionally, when looking backwards multiple steps of reward for this valuation, it is possible (and likely) that the reward is diminished by a certain factor that grows over time. In this manner the cumulative value of the valuation is weighted in proportion to how long ago that action was taken. It is convenient to state this concisely, but there is much to be considered in trying to select this next action optimally. Given a deterministic environment that has been fully explored choosing the a' is distinct and simple. Each state s would have an optimal action a that maps to the highest valued option, thus the optimal action. In either non-deterministic or probabilistic environments these evaluations must be performed using the underlying probabilities or experimental values thus far (Equation 1.6). If the full ramifications of all choices cannot be determined at the time of evaluation, or if doing so would take inordinate time, then a heuristic must be used. These heuristics may range from scores or values to expected values and rewards. In this environment the a' is

determined by choosing the maximum value of the available choices (Equation 1.7). Here $E(s, a)$ is the expected value based on experience thus far of taking action a from state s.

$$\pi : S \rightarrow A, where : \forall s \in S, s \mapsto a', where : a' \rightarrow max \tag{1.3}$$

$$a' \rightarrow max : \underset{a}{\operatorname{argmax}} V_a(s_t) \tag{1.4}$$

$$V_a(s_t) = r(s_t) \tag{1.5}$$

$$a' \rightarrow max : \underset{a}{\operatorname{argmax}} P(s|a)V_a(s_t) \tag{1.6}$$

$$a' \rightarrow max : \underset{a}{\operatorname{argmax}} E(s, a) \tag{1.7}$$

These values can be determined through machine learning, experience, or domain knowledge. In fact, this determination can be made using more than one of these techniques as more experience is gained in the current trial. When the agent has no prior knowledge they must make their choices stochastically to explore each option and track their rewards for taking each action. Once enough states have been visited and actions taken, the exploration (i.e., favoring stochastic selection to maximize

testing each option) transitions to exploitation (i.e., taking the best actions seen thus far to maximize reward). Once these most-favorable choices ($w.r.t.$ reward or utility) have been learned, they can be saved and re-used. This forms a policy that the agent can now start from and not have to relearn an environment. This is assuming that either the states stay the same or are only marginally different from the states in the learning environment. Additionally, it is not necessary to assume a fully exploitation-oriented approach. The agent can take the favorable action from its experience 90% of the time and make stochastic choices the other 10% of the time to make sure there is not a better choice. Once this policy π has been learned, it can be added to the set of policies Π to form a collection of policies. The agent or set of agents can then share this collection of policies as prior experience for each of them. Selecting from among these policies is the domain of strategy.

A strategy, σ, selects the optimal (or desired) policy, π, from a subset of policies drawn from the set of all policies, Π, to accomplish a goal, g, or meta-set of goals, G (Equation 1.8). All σ are in Σ (Equation 1.9), the set of all strategies. As shown in Equation 1.10, the set of all strategies Σ is the powerset of all possible policies Π. As seen generically in Equation 1.11, this selection is simply putting in force the best policy given the current set of states and actions. The method for such selection, however, is not so easily contrived. There is a similar notion of expectation that will be used. This expectation function will examine the actions $a \in A$ taken by the current policy π in force and by those of each other known policy $\pi \in \Pi$. The $argmax$ of these values is then taken, and this may indicate that either the current policy π

is the best policy to have in place or there is a shift in policy that needs to occur. Further, there is a threshold ϵ that is considered before a policy change occurs in order to provide some momentum to the current policy and avoid unnecessary vacillations in policy change (Equation 1.12). If the δ between the current π and π' is less than ϵ, the current policy, π, stays in place; otherwise, a change is initiated. This mechanism and relevant valuations are considered within the experiments.

$$\pi \in \sigma \subseteq \Pi \tag{1.8}$$

$$\sigma \in \Sigma \tag{1.9}$$

$$\Sigma = \mathcal{P}(\Pi) \tag{1.10}$$

$$\sigma = \{\pi_0, \pi_1, ..., \pi_n\}, where : n = |\sigma| \tag{1.11}$$

$$\sigma : \underset{\pi}{\operatorname{argmax}}\, E^{\pi}(s, a) - \epsilon, \forall \pi \in \sigma \rightarrow \pi' \tag{1.12}$$

1.10.4 Strategy Example

Consider the multi-team strategies of imitation and optimization. While there are many strategies that can be described using these methods, the focus will start on two

such examples. The first is the optimizer strategy which seeks to perform optimally at all steps. This agent will be given domain knowledge to ensure that it does make the optimal choice and thus will not represent an actual learned type, but rather form a control group. The second type considered is the imitator. The imitator attempts to mimic the behavior of the optimizers who, from its perspective, seem to understand the underlying FST's that govern the game. This optimizer is playing the role of a surrogate in the game for the environment. Using this special access to domain knowledge it is imitating (or approximating) what the environment knows as truth. While this is unrealistic for any normal player, it provides a ground truth and a differential for comparison to understand the action of the imitator.

The imitator strategy, s_i, is described by a tuple, $s_i = (M, \pi, \mu, \delta, m_0)$. Here, M is the finite set denoting the memory of the strategy, $m_0 \in M$ is the initial memory, $\delta : A \times M \to M$ is the memory update function, π is the policy that the player is currently following where $v \in V$ is enabled at v, and $\mu : M \to [n]$ is the imitation map. This imitator strategy is then selecting the next move, or the next policy to put in place, for the given player, i. This imitator must create and maintain a data structure of memory and a belief network about the other players being monitored. This data structure is essentially an additional move memory, just for different players.

Definition 3 *Given a game $\mathcal{G}$, with n players, and a player, i, with a strategy, σ_i, be defined as an imitator $(M, \pi, \mu, \delta, m_0)$. Adding memory for other players transforms m_0 into a vector, $m[n+1]$ with the first entry, $m[0]$ being the memory of the player and all other entries, $m[n]$, being the memory holding the moves of the*

other n players. Additionally, and without loss of generality, the notion of teams can be introduced by making $m[n+1]$ *multidimensional. Further, let* $\beta[n]$ *be a belief network which holds a success score for each of the n players. This constantly evaluated function maintains its belief as to the best player, and makes them the target for imitation. The moves that need to be evaluated (imitated) are already stored in* $m[n+1]$*. This results in the new strategy definition for imitation being* $s_i = (M, \pi, \mu, \delta, m[n+1], \beta[n])$.

This imitation strategy can also be represented as a FST, $\mathcal{R} = (M', \delta', g', m_I)$ which, given the current state of the game, the extant memory data structure, and the belief network, produces the next move for the player. This next move will be based on the best measured move from the history (memory) of the observed best player. Strategy is then the selection of an optimal policy from among the available policies based in the observed environment variables (e.g., gamestate, score, timing, etc.).

1.11 Conclusion

Understanding this background may help to inform the process by which the current specification of strategy and the current implementation of strategy-based systems, and as such it was offered here. Building from this, the research can proceed to the new system created from this background research, presented in Chapter 3.

Chapter 2

Related Works

There has been some work in the theoretical side, based on the mathematical model for strategy interaction and selection by (Soumya Paul and R. Ramanujam, 2010) concerning the use and impact of imitation in large games. This paper lays the mathematical and theoretical foundation upon which the proposed research builds - namely the formation of an arena within which a game can be played that shows the ability to imitate the strategies of optimal performers. This game is seen, then, as a microcosm of various real-world competitive environments.

The research proposed herein will take this theoretical work into the real worlds with practical robotic models and also add teams into the mix. This premise is expanded upon in (Soumya Paul and R. Ramanujam, 2011a) which adds the concept of interactivity on strategy and learning, casting this interactivity as a 'society' with associated rules. This paper will be used in the research proposed to further the examination of team behavior to create an environment (i.e., the society) within which these teams can compete or cooperate. This societal rule set will then be shown to create governing precepts for all agents, and correspondingly for all teams. This

societal influence can also be seen as an embedded element of any and all strategies that are implemented within this framework. This may also form what may be termed a base-line strategy upon which individual strategies are built, or a super-strategy to which all other strategies must conform.

These same researchers furthered their work in (Soumya Paul and R. Ramanujam, 2011b), suggesting that there is a way to expand this work to larger systems by breaking down the entire system of interactions into 'neighborhoods' where they are considered locally. This will be shown in this proposed research to form a functional decomposition of even larger systems into simpler elements that still accurately model the same behaviors, strategies, and outcomes. This proposal, therefore, seeks to expand on the combination of these ideas and move them from the theoretical to the practical, adding them to prior research. While these papers represent the bulk of the previous work that will be built upon, there are several others that have provided background material relevant to this research effort.

(Simon Parson and Michael Woolridge, 2002) provides a general background in both Game Theory and Decision Theory, specifically as it applies to multi-agent systems. (Michael Bowling and Manuela Veloso, 2002) introduces Game Theory into multi-agent learning. These works give several approaches to solving multi-agent learning systems and their mathematical foundations. These reference works provide the underpinning of the work that will be introduced herein in multi-agent systems and large scale game solutions.

There is much foundational work in both game theory and learning in multi-agent systems. Rather than review each of the multitudinous examples (like (Littman, 1994),(Hu et al., 1998), and (Greenwald et al., 2003)) in this proposal, there is a larger work that summarizes each of these and compares them. (Bowling et al., 2004) also firmly establishes this background while entrenching itself in the multi-agent learning scenario, and in particular in how the related work from game theory (e.g., the Nash equilibrium) fits into the more limiting field of multi-agent learning. This paper lays the mathematical background for moving the oft-cited Nash-equilibrium into the realm of stochastic games with multiple players. In a Nash-equilibrium each player is performing optimally and no other player can do better than they are. In limited games, where the intentions of all players must be considered and not all actions are available at all times to all players, there is not guaranteed to be such an equilibrium. The authors propose that, by understanding the limitation of the players and the interaction of their joint policies, restricted equilibria can be found. These are formulated as Best-Response learners. This leads the authors to consider the implicit games that arise within the explicit game itself. These sub-games are often simpler and more reliably arrive at equilibria. Thus these learners can often lead to insights about how the game will be played in this multi-agent simulation or even to the limits that need to be in place so that a Nash-equilibrium can be found. More importantly to this proposal, the concept of needing multiple policies, interacting policies, and strategies is introduced and discussed. This paper offers that learning in a multi-agent environment may not even be possible without such super-policies (or

strategies). Additionally, there is much insight offered into the framing of the game to ensure that it is such a stochastic game that learning can, indeed, occur. This served as a check for the formulation of the stochastic game introduced in this research so that individual agents can exhibit behaviors that lead to the inference of their own behaviors, and subsequently lead to the inference of the team strategy. Without this mathematical foundation and exemplary work to stand on, this proposal would be weighed down with many more proofs and theorems. Instead, this work utilizes these well-formed ideas and build on them.

In another paper from Kwun Han and Manuela Veloso, (Han et al., 2000), the authors present a similar system to what is proposed in this paper. The authors propose a system that can detect the behaviors of robots within the environment via their vision system. By using Hidden Markov Models (HMMs), which they recast as Behavioral Hidden Markov Models (BHMMs), to represent robot behaviors they construct probabilistic choices as to which behavior the robot is exhibiting. They present this modeling and recognition using HMMs to be novel, especially in hoping to use this in a multi-agent system. Their approach is to utilize the data coming from their global overhead vision system to feed their behavior membership decision. One robot is trying to infer the strategic behavior from the other by comparing the actions being observed with a known set of behaviors. Their goal with this system is to automate the 'play-by-play' narration of the game. One important insight they bring is that complex behaviors can be broken down into discrete stages that are much simpler. These stages are what is represented by the BHMMs. The models

have several states: initial states that mark the beginning of the behavior stage, accept states that represent the completion of an action, immediate states that show progress through the behavior, and reject states that mean that the model has not recognized or matched a known behavior. By marking the models progress through these stages the intention is to arrive at an accepted state that gives the action being exhibited by the robot being observed. This process involves an observation feature extraction based on the location and orientation of the objects within the field of view, in this case the robot player on the soccer field and its relation to a ball.

Although they set out to show this system working in a multi-agent arena, they quickly note that their system and experiments are based on only one agent in view. The work proposed herein looks to not only recognize the behaviors of individual multiple agents but to additionally aggregate these individual behaviors into a team strategy. One issue that they ran into in their research was the timing of 'catching' these actions because of the constant movement through these states. If the recognizer was not launched at the correct time is is unlikely to recognize the behavior. It needs to be instantiated close to the start of the behavior. To increase the odds of catching the behaviors they instantiated new recognizers at regular intervals. In the work proposed in this paper this is overcome by using graphical models with multiple entry and exit points. Using such a system of models allows for any actionable feature that can be extracted to find a place (or, likely, multiple places) to enter, traverse, or exit a model. This overcomes the issues with having to launch multiple high-resource recognizers that may or may not catch behaviors.

A paper from Chernova and Veloso, (Chernova and Veloso, 2009), introduces some unique methods of segmenting the behaviors being observed so that recognition (and, consequently learning) can be more tractable. While this work is not as closely related as others, as it deals with human agents interacting with computer agents in the context of repetitive learning with correction, it does speak to the need to bootstrap the policy / behavior learning and to reinforce that with examples. This idea is incorporated into the belief network that establishes that predominantly probable strategy that is being used by another team in the game scenario. Another paper, (Fernández et al., 2010), adds Probabilistic Policy Reuse (first introduced in (Fernández and Veloso, 2005) and expanded in (Fernández and Veloso, 2006) to reinforcement learning) to inter-task transfer learning. This specific applications is not a perfect match, but the ideas of transfer learning and storing policies for reuse do provide insight into how this proposal segments and stores strategies (or the composite pieces of behaviors). In particular there is good insight into the selection process for which policy would be best to reuse. Additional insight was provided in how the authors framed the tuple construction from the standard MDP to provide additional information in separate tuples that comprise the domain and task models.

In a related paper, (Bowling and Veloso, 2001), the authors lay their foundational work in the mixed-policy approach to finding equilibria (perhaps even a Nash equilibrium) in stochastic games. They argue that to learn in stochastic games, themselves an extension of MDP's into a multi-agent environment, requires two properties to be fulfilled: rationality (the idea that their focus agent will narrow in on

an optimal solution if the other agents strategy remains stationary) and convergence (that all players will converge to stationary policies). They summarize the findings of other approaches to solving stochastic games in a multi-agent environment and introduce their own solution. They first looked at Single-Agent Learners that pursue their own optimal solution and treat other agents as part of the environment, Joint Action Learners that choose their actions by assuming that the others agents policies are stationary and estimable, and Minimax-Q which observes both actions and rewards in an effort to search explicitly for an equilibrium. They then introduce WoLF Policy Hill-Climbing as the only known algorithm that can satisfy both the rationality and convergence requirements that none of the other three algorithms could. This technique uses their variable learning rate (introduced in (Michael Bowling and Manuela Veloso, 2002)) to learn more rapidly when losing and less so when winning (WoLF is 'win or learn fast'). This adaptation of Policy Hill-Climbing (which is based on Q-learning) fluctuates the learning rate to create a faster and more rational convergence to an optimal policy. They present the results of this policy in several stochastic game scenarios. This paper provides insight into stochastic games, their restrictions and sufficient conditions, and the problems associated with learning in this environment. While this scenario and approach align in their theory with the this proposal, they differ in technique as this proposal asserts that it is also possible to address these constraints (namely rationality and convergence) through a probabilistic search of a graphical model of strategies.

An excellent treatise on team management, role assignment, and in-game communication can be found in (Stone and Veloso, 1999). This work gives a good design principles and framework information on exactly the type of scenario envisioned by this proposal - multi-agent team coordinated behavior with both cooperative and adversarial elements. This paper provides much of the initial material for considering teamwork inside of the stochastic game presented in this proposal and in the communication sections of the multi-agent and strategy inference portions.

A little further back, in the mid-1990's, there was a conference that summarized the state-of-the-art in reinforcement learning in multi-agent environments. The proceedings of that conference, (Weiss and Sen, 1995), provide an appropriate starting point for this line of research. Mataric et al. then took several passes through this material, including (Matarić, 1996) and the more relevant (Matarić, 1997). In this latter work the themes revolve around adapting reinforcement learning to fit in multi-robot scenarios. The two main issues that have to be overcome are the prohibitively large state space and the credit assignment problem. Their work is behavior based, so that helps to consolidate some of the multiple action steps into a single step. By exchanging conditions for states and behaviors for actions they have thus reduced their search space significantly (what they refer to as 'shaped reinforcement'). While these findings and results are not used directly, the idea of the compression of the overall search space does appear in the hierarchical approach of the state-action to behaviors to strategies to intelligence model. It should also be noted that this work is supported experimentally with real robots performing a legitimate task. This

separates this work from many others that are only theoretical or simulation-based. Also, this work treats the team goals as a single goal, with each member performing the same work to achieve this goal. In the research proposed herein, the teams are made up of individuals with their own goals and team goals, and the larger strategy applies to both individuals and teams. There was an additional paper in this space that summarizes the state-of-the-art again, This paper, (Yang and Gu, 2004), brings these other papers and several more together to summarize the field. While this is informative, it does not add insight to this proposal.

The work of Vazirani (Vazirani, 1989) described the complexity classes of finding complete or perfect matchings in variations of complete graphs. It documents how to decompose the problem of $K_{3,3}$ into subsets of examination using K_5 discrete elements. This formulation decomposes that problem from the complexities of intractability into the tractable realm. Namely, considering the decomposition as a set of parallel sub-optimizations that can be discretely considered provides a lower-NP bound, though it requires parallel processing. Further, the paper introduces the exponential increase in complexity when considering homeomorphic graphs. The insight of graph decomposition is utilized herein to inform the process of culling the list of viable candidate trees and the further intractability of isomorphic graphs is confirmed.

In (Das et al., 2013), the authors investigate paths and matchings within k-trees. They analyze the complexity classes of such matching and searches in these trees. Their experiments substantiate the claims of the complexity of matching trees and offer motivation for approximate solutions to this class of problems. Their insight

into tree-decomposition is also helpful in organizing solutions to large-scale problems in the realm of tree-matching. Their work stops short of larger scale graphs and does not consider approximate solutions to the more intractable issues of homeomorphism and isomorphism.

Datta, et al., study the effects of moving the graph matching problem into the area of bipartite graphs. In (Datta et al., 2010) they established an algorithm to find matches within and between graphs and analyzed the relevant complexities, assigning simplex matching to NC. This lays the groundwork herein where the limits of this complexity were pushed into higher bounds by matching homeomorphic and isomorphic graphs. Their work also helped to inform the baseline algorithm used herein for comparison to the approximate solution (along with several other works and the author's own research).

The work of Fukuda, et al., further explores the complexity of matchings within bipartite graphs and aims to make improvements to the process. Their work (Fukuda and Matsui, 1994) elucidates the difficulties and complexities of such matchings. In particular, they move from $O((c+I)n^2)$ to $O(c(n+m)+n^{2.5})$. While they do not deal directly with homeomorphic and isomorphic matchings, they do further describe the relationship with complexity and memory management. The algorithm they propose does, in fact, lower the computational complexity, but at the cost of increased memory utilization. Additionally, they recognize that computing such difficult matchings pushes the limits of computability. As a result, herein, the entire problem of doing

these matchings in real-time is further clarified as intractable and in need of the approximate solution proposed in this research.

The work of Weber and Mateas (Weber and Mateas, 2009), presents a strategy prediction technique for a RTS game (Starcraft). Their work uses logs of previous plays to analyze specific strategies and begin to predict which of them was being utilized by an opponent. Their approach is along the same lines as what SiMAMT and the SIE are doing, but they are using an existing turn-based game rather than their own simulation. The game they are studying has single players, but we wish to analyze strategic interactions at a more complicated scale that include multi-agent teams and multiple teams within the environment.

In the work of Laviers et al. (Laviers et al., 2009), the authors seek to make an alternative play based on reading the opponent's previous formations and predicting their current play. If the current play they recognize is predicted to outperform their own play, they attempt to make the change to a better play in real-time. Their work is not multi-agent in that it considers the play itself and not the individual actions of the players, nor is it multi-team as they are only considering one team (namely, the opponent). Their procedure and overall idea is very well done and informative for our work.

These and other related works have been examined and interwoven into this research.

Chapter 3

Strategy and Strategy-Based Specification

3.1 Introduction and Overview

With the introduction of strategy already offered and the related works in mind, we propose the rationale for and the implemented solution to the issues of implementing intelligent strategic behavior in large-scale and highly-complex multi-agent multi-team environments. The area of predictive computing, attempting to infer from current actions what the next action may be, is dominated by Markovian processes like Markov Decision Processes (MDPs) and Partially-Observable MDPs (POMDPs). While these processes can look back a few steps, the general indication is that this is of little importance and they rarely end up looking back more than one step. While this research does not attempt to de-emphasize the Markov property or this body of work, it does propose to take a broader look at sets of actions, larger action sets, and a more hierarchical approach to understanding the next likely action of individual agents within a system. Further, it proposes to take the interactions of multiple agents within a single system out of these more classical models and into an arena of higher reasoning that involves game theory and strategies. *This research contributes*

the notion of strategy inference by coalescing action sets into larger models that understand cooperating and competing agents, teams, and interactive time strategic interactions. Often it is intractable, due to the exponentially increasing computation involved, to consider all previous actions. This lead to the research that introduced the Markov property. This research chooses a different tact, to track the actions of the agents as member elements of a Probabilistic Graphical Model (PGM). As the actions are tracked and matched to existing strategic models, a Belief Network (BN) is produced that calculates the most likely candidate for the current strategy being implemented by the agents in the interaction. *This will help the focus agent answer two important questions: what strategy are the other agents following and how are they choosing which strategy to follow?* With this information the focus agent can consider its own next action with the full-information of the most likely next move of other agents. This interaction — the agent with its environment, the agent within its own team, the agent with other agents — should be able to operate in interactive time to allow multi-agent strategic interaction since it will be tracking but not logging all actions.

3.2 Strategy Representation

It is essential to establish a method of representing a strategy in order for an agent to be able to implement it. Further, it is essential that this representation allow comparisons among strategies (e.g., inferring which strategy is in use through observation). Finally, this representation must allow for real-valued evaluation to

determine which strategy (or policy within the strategy) is the best choice for the current environment. With these representations in place, it will be possible for the learned policies of an agent to guide the agent along a larger path to 'win' more complicated or dynamic games. It is also possible, given this ability to reason about the strategy of another agent, to determine if that agent is cooperating or competing with them without *a priori* knowledge. Once these various strategies are understood, any agent, or team of agents, can then evaluate their strategy with respect to others and decide if they should keep their current strategy, adapt their current strategy, or replace their current strategy in favor of this better one. This is the reality of how complex games are played in the real-world, and how large scale models interact (e.g., stock markets, competition among businesses, commodities trading). It is also clear that game theory will help in this evaluation of strategies to better model the interactions of both team members and opponents. Game Theory and Decision Theory will be essential in determining which agents are cooperating and which are competing, how well each agent is performing, and how the focus agent should react to this knowledge. This will be discussed in Chapter 4, but for now these strategies are mapped as patterns that are unique.

3.3 Strategy Inference

The core issue with strategy is the ability to model, infer, and adapt strategies based on the environment and the interactions with the other agents. The ability to recognize the behaviors of other agents is increasingly needed as more multi-agent

intelligent systems emerge (Han et al., 2000). As previously mentioned, the first issue is to model the strategies uniquely using patterns. Once each strategy has been modeled the system is trained to recognize them. One example of strategic modeling is that used by the Overwatch system, introduced and described in Chapter 6. In Overwatch the centralized system holds the models, but bootstrap learning 'teaches' each agent the various strategies by imprinting these patterns. From here, the individual agents can then perform a strategy pattern recognition and matching algorithm to match the observed actions of other agents with its known strategy patterns. As noted in (Han et al., 2000), this allows for the focus agent to adapt its strategy based on these observations. This probabilistic matching uses a belief threshold to identify which of the patterns is most likely in place and then uses this belief to process its own action set (based on its current policy, which is derived from its strategy). Using this probabilistic matching methodology, each agent can identify its teammates, its opponents, and those agents who are either neutral or are yet to be identified. Though this probabilistic matching mechanism will be discussed later in Chapter 4, it is important to note here that for n strategies known (that have been learned or provided), there are n+1 strategy patterns considered for each other agent in the system. The n strategies represent the known strategies while the $(n+1)^{th}$ strategy is a placeholder for any unknown strategy. This allows the focus agent to decide that it does not recognize the strategy in play and it can record this new strategy for future use. This is the belief that the focus agent is witnessing a strategy that it has not yet learned. When an agent is seeing an opposing strategy for the

first time it must continue to evaluate its own strategy (and any other strategies it has access to) for performance. This strategy inference, matching observed actions with know strategy models, is fundamental to learning within strategic environments. This observation of the action sets of other agents helps the focus agent determine the best strategy (i.e., the strategy that maximizes the performance of the agent or team of agents) that they should be following. The following quote offers an early mandate for agents within multi-agent interactive systems:

> "...an agent...must (a) recognize that there are other agents, (b) compute some of the other agent's possible plans, (c) compute how the other agent's plans interact with its own plans, and (d) decide on the best action in view of these interactions. So [both competition and cooperation] require a model of the other agent's plans." (Russell and Norvig, 2003)

The concept of strategy inference is introduced here, but the complete specification for it and its implementation are discussed in detail in the SiMAMT Framework, Chapter 4.

3.4 Strategy Application

Strategy models, as have been described, are essential in both the execution of a strategy and the recognition of strategies in others. This is proposed herein as a belief network that matches actions to models of strategy. Once the focus agent has formed its beliefs about the other agent's strategies it can then compare the rewards

that the other agents would be getting for such a strategy and decide how, if at all, it should adapt its own strategy. If the strategies it believes are being used by other teams of agents are getting a higher reward than its current strategy, it can either adapt its strategy to mimic that one or adopt that agent's or team of agents' strategy as its own. In machine learning, the reward is the output of the action function or the sum total of a set of action functions. The reward can be positive for good actions, negative for poor actions, or zero for neutral actions. Thus, if the strategy of another agent is gathering less reward (as calculated by an evaluation function or fitness function), the focus agent will keep it current strategy. Alternatively, in a team environment, if the agent being considered is on the same team, the focus agent can communicate an update to that teammate. In this manner, the agents can form a better team (i.e., one where the team's overall reward is higher) or learn to imitate the behavior of other agents, or teams of agents, in the system. This type of imitation, whose theoretical foundation is laid out in (Soumya Paul and R. Ramanujam, 2010), can be a powerful way for a team with fewer members or fewer resources to perform at a higher level than if they had to learn on their own through long trials or historical data. There are several advantages to be gained with such an ability. Teams can leverage the power of distributed learning by communicating their most effective strategies. Opponents can ensure maximal success by considering both their own rewards (and their individual learning) and the rewards of the the strategies other teams are following. The entire system can coordinate the work of diversely skilled teams being utilized to their own individual maximum and thereby achieve

a higher overall reward through its synergism (i.e., as each agent within the team learns which policies are producing the best reward they can share this information with their teammates). Each of these scenarios, mentioned in this section, shows how widespread the application of strategy inference can be and how this type of learning and operation can achieve a higher overall reward than the state of the art systems, as will be shown in the experimentation of the system.

3.5 Strategy-based Models

The foundational work, presented earlier as background and history, shows the application from the general perspective and applies to a single agent. This means that the strategy, in the single-agent context, is monitoring and adapting the quality of the policy the agent is currently using. When we move to multi-agent teams we shift into the higher level of strategic thinking. In the team concept, the behavior is the collection of actions the agent will take. A policy maps a particular agent to a behavior. However, when there are teams of agents, there is a need for a larger policy for the team, a super-policy. This super-policy is called a strategy, and it represents a mapping of each member of the team of agents with their respective policy (that, in turn, maps each agent to a behavior). Finally, we utilize a new engine to decide which strategy should be in place for the team (and switch it if necessary). This process is elaborated on and formalized below.

As a historical note, strategy has been applied to various battlefields throughout history. It is generally thought of, as defined formally in the introduction, as the

coordination of movements of units in the field of battle. The theater-wide view of strategy in multi-layered. At the highest level, the strategy should tell the teams (units) of agents where to go, how to get there, and what their goal is. One layer down, each unit is following a strategy that is prescribed to it by the higher-level strategy and seeks to guide each sub-unit into their respective positions, formations, and goals. Each agent within the unit, then, is mapped to a policy by the team strategy. The agents are then guided by these policies, each of which is a mapping of a behavior to a particular agent (here, a soldier). Each agent is then moved, according to their behavior (i.e., a series of movements with speed and probabilities) from position to position. Each of these positional assignments included in the behavior tells the agent exactly where to be (a location), and their posture at that location (prone, standing, charging, etc.). The historical battlefield plan shown in Figure 3.1 offers a glimpse of the overall strategy of the battle. Of course, it does not offer the detail that the strategies proposed do, but it does lead into the discussion about movements throughout a scenario and how they can be determined, derived, and assigned.

3.6 Strategy-Based Systems Specification

SiMAMT is a framework for strategy-based multi-agent multi-team systems. As indicated previously, a multi-agent team is a group of agents that are working together as one cohesive unit (meaning that they are individuals but share a common group goal). This contrasts with systems where there are multiple agents that each act independently (meaning each agent pursues their own goal without consideration to

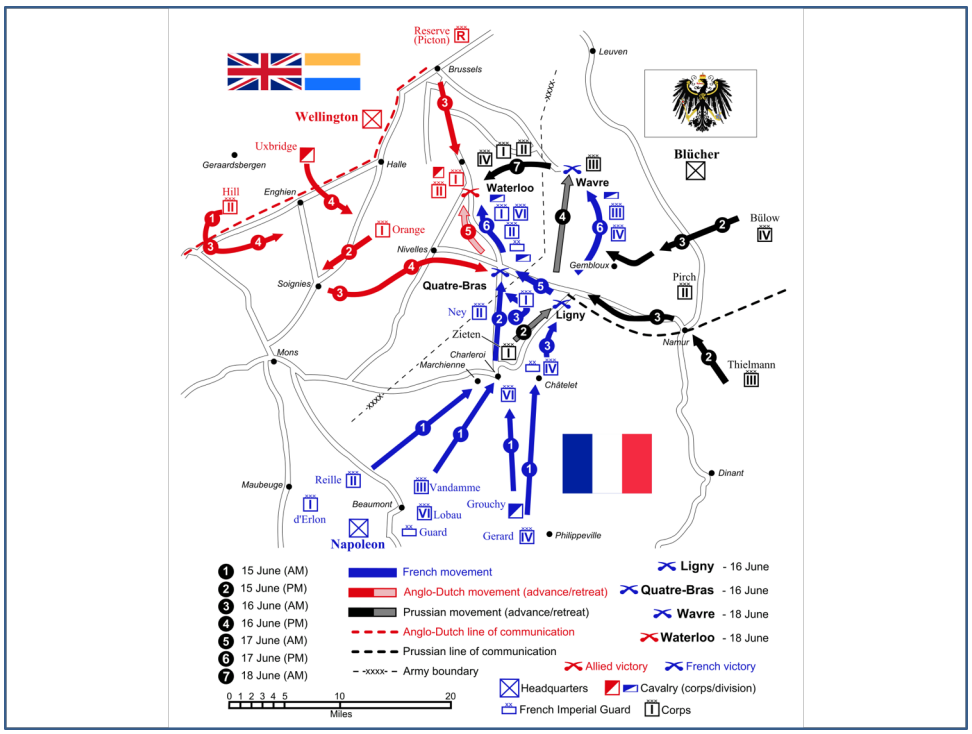


Figure 3.1: The Battle of Waterloo Strategic Overview

group objectives). A multi-team scenario is a system wherein multiple cooperating groups of agents (teams) are competing against other such groups. This is not exclusive - for example, there could be a total of nine teams, but they work together in groups of three each, resulting in three-team squads that are cooperating while the three squads are competing. Or, similarly, there could be five alliances, each comprised of a different number of teams (e.g., Alliance 1 has 3 teams, Alliance 2 has 4 teams, Alliance 3 has 6 teams, Alliance 4 has 3 teams, Alliance 5 has 4 teams; Alliance 1, 2, and 3 are cooperating and opposing the Super-Alliance of 4 and 5). There are many combinatorial possibilities for the alignments (i.e., teams allied in cooperation or standing in opposition), all of which are supported by the SiMAMT framework.

In the general case the strategy should consider a number of elements. From the highest layer of abstraction to the physical representation we wish to build a hierarchical view of the system. To that end, the highest level of abstraction is the simulation itself. This may take the form of a proper simulation (modeling behaviors within a system), a game (modeling interaction between agents or teams of agents), or a model (modeling the system itself). The simulation, via the Intelligent Strategy Selection Engine (ISSE), assigns a strategy to each team of agents (defining the role for each member of the team). Each strategy then uses policies (e.g., CoverFire, Defend the Goal, Conserve Fuel, etc.) to map a behavior to each agent for which it is responsible. Each strategy maps from the set of policies from which it can choose (this set is a subset of the total set of policies) to each team member. Each behavior, mapped by the policy to the agent, is providing the action sequence and transition function for a particular agent. The agent represents the physical object being modeled within the system.

One critical element to understanding and deploying strategy within a simulated environment is the Movement Dependency Diagram (MDD). The MDD is a diagram that is the result of a search through the entire state space (in each system, states are defined by that system as locations, positions, or situations where an agent is located and from which they can take actions to shift their state). Generically, the state space of an environment is the list of connected states that are reachable through every possible action. In strategic simulations this is the entire list of movements - that is, any and every action that moves an agent from one position to another.

This creates a diagram that shows all possible movements and all possible subsequent movements from those, thus creating a Total Movement Dependency Diagram. The Total Movement Dependency Diagram can be made into sub-graphs where each sub-graph contains a particular set of connected moves within the larger set of moves. These sub-graphs can then be tied to certain strategic behavior (e.g., playing a particular position in soccer). For examples of this, see Chapter 5, where this concept is presented in full.

The agent, o, is represented by a tuple, $o(\phi, \psi)$, where ϕ represents the limitations placed on the agent by the simulation and ψ represents the performance variables of the agent (Equation 3.1). These limitations might be physical (height, weight, supplies quantity, resources usable, etc.) or virtual (amount of memory available, number of processing cores, etc.), but in the general sense they indicate the constraints placed on the individual player by their environment. The performance variables are the specific attributes of the agent and indicate how they impact their environment. These might include their speed of movement, their accuracy, their willingness to cooperate, or any elements that dictate the 'personality' of the agent. The set of all agents within a system is O, so $o \in O$.

$$o =< \phi, \psi >, o \in O \tag{3.1}$$

The field upon which the simulation takes place is divided into positions that an agent can occupy. Each position, p, is a tuple, $p(l, \rho)$, where l represents the location

on the field (referenced to a mapping of the field) and ρ represents the posture of the agent at that location (Equation 3.2). For example, posture could be *left-side*, *undercover*, *below*, *behind*, etc. In each case, posture represents the alignment (or orientation) of the agent with the location on the field and the objects within the field (as illustrated in Figure 3.2). Each position, p, is drawn from the set of all positions, P, such that $p \in P$.

$$p =< l, \rho >, p \in P \tag{3.2}$$

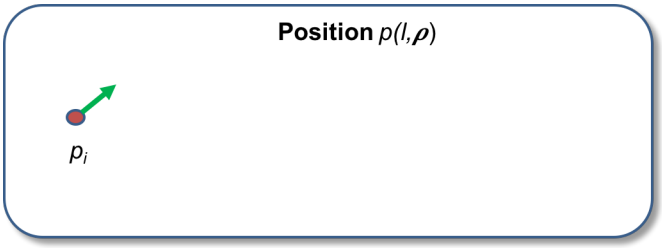


Figure 3.2: Position Diagram

These positions comprise the base element that is the state, s. The state can be atomic or composite, but it is the lowest level of the overall hierarchy of elemental considerations. In many systems the state may be a position description defined by characteristics such as an ID number of the position (p_n), the name of the position (p_{name}), and the position itself (p_i)(Equation 3.3, Figure 3.3). Such an encoding encapsulates the field elements within it by position number and specifies the posture of the agent at that position (e.g., standing, prone, etc.) or where the agent could be located within that position relative to any obstacle (e.g., left, under cover, right

of obstacle). This positional and orientational encoding can also be used to define aspects of the field (e.g., other positions on the field the agent can see for observation of movement or firing on other agents). The partial observability of the field for each agent is defined and determined by these states. Thus, the state can be utilized to describe a wealth of information for the agents within the simulation.

$$s =< p_n, p_{name}, p_i >, s \in S \tag{3.3}$$

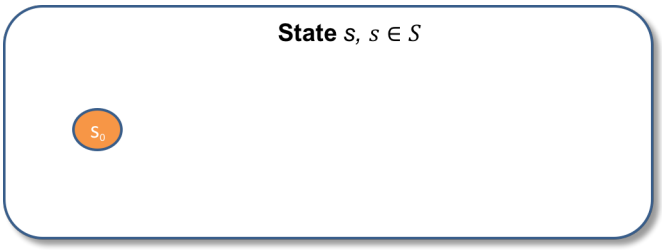


Figure 3.3: State Diagram

These states can then be encapsulated inside of movements. A movement, m, is defined by Equation 3.4, and illustrated in Figure 3.4. Accordingly, a movement is a mapping from the set of states to the set of states, meaning that each move is initiated from one state in the state space and ends on another state in the state space (even if that state is the current state). Here, given a next position (that is, the position encapsulated within the next state), p', m_s is the speed of movement of the agent towards this position, and m_p is the probability of making such a move (these probabilities are either provided by the strategy-based modeling or learned as the simulation progresses). The intent is to capture the agent's movements throughout

the field in discernible elements. In practice, these variables are modified by the agent's individual performance variables (these variables are part of the profile for each agent and represent the innate characteristics of the agent — how fast they are, how aggressive they are, etc.). For example, a movement $m_5(201, 4, 0.25)$ would be a proposed move from the current position to position 201 with a speed of 4 and a probability of 0.25. If a certain agent, for example o_3, has a speed modification (one of their performance variables) of 0.5, then this movement, if initiated, would be done at a speed of 2 (rather than 4). Similarly, if this same agent had a move likelihood value of 0.75 (again, another performance variable), then the likelihood of their movement is now 0.1875 (0.25 x 0.75). So, in this case, agent o_3 would make movement m_5 to position 201 with a speed of 2 and a likelihood of 0.1875. These variables show how the individuality of the agent plays into the simulation of their behavior and how their 'personality' impacts the performance of the team.

$$m : S \rightarrow S, \text{subject to } p', m_s, m_p \tag{3.4}$$

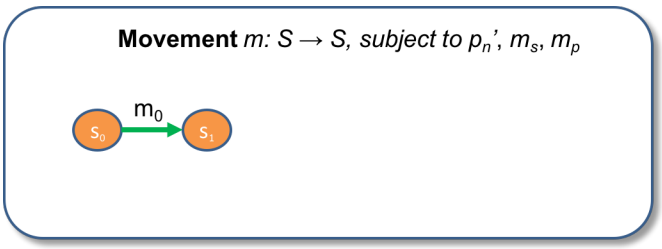


Figure 3.4: Movement Diagram

These movements can be joined together to form a behavior. This particular behavior is a charting of movements from state to state throughout the simulation space. A behavior is represented by a non-deterministic finite state transducer (NFST). An NFST is defined generically by the tuple $(\Sigma, \Gamma, S, s_0, \delta, \omega, F)$ (Equation 3.5), where Σ represents the input alphabet (the data provided as input for processing within the NFST, designed in a Turing tape paradigm), Γ is the output alphabet (the data output from the NFST, also designed as a Turing tape), S the set of all possible states, s_0 is the initial state, δ is the transition function (using the input alphabet to perform a mapping with the set of states to determine the next state), ω is the output function (using the input alphabet and the set of states to provide the output of state transitions), and F is the set of final states (accept states, the subset of states which represent terminal states within the NFST) ((Mohri et al., 2000), (Moore, 1971)). NFSTs are suitable for such behavior modeling as they emulate the procedural patterns of real systems ((Rabin and Scott, 1959), (Moore, 1956)).

$$b =< \Sigma, \Gamma, S, s_0, \delta, \omega, F > \tag{3.5}$$

For behaviors in this system, however, the NFST definition is modified to match the variables within the system. The result is found in Equation 3.6 and illustrated in Figure 3.5. The equation shows the formalization of the NFST and the relevant input, output, and transitions. The figure shows an example behavior vector that illustrates the agent transitions from state to state. The solid lines indicate definitive

(deterministic) next states while the dashed arrows indicate probable next states (non-deterministic). An agent can move along any path within this behavior vector.

$$b =< M, T, S, s_0, \delta, \omega, F >, b \in B \tag{3.6}$$

where

- M is the input set (set of all possible movements)
- T is the output set (set of all state transitions and observations from the agents)
- S is the set of states (set of all possible positions)
- s_0 is the initial position of the agent ($s_0 \in S$)
- δ is the transition function ($\delta : S \times (M \cup \{\varepsilon\}) \rightarrow \mathcal{P}(S)$)
- ω is the output function ($\omega : S \times (M \cup \{\varepsilon\}) \times S \rightarrow T^*$)
- F is the set of conditions of completion for the agent

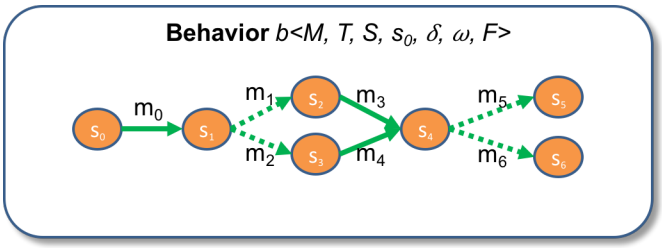


Figure 3.5: Behavior Diagram

The cited works show that many systems can be modeled using NFSTs. These systems take in an input (a string of symbols defined by the system) and produce

a translation of these input symbols into output symbols (also a string of symbols defined by the system). As noted, this can be defined in terms of a Turing machine with two tapes. One tape, the input tape, is covered in symbols from the input alphabet. The NFST, as a Turing machine, reads in these symbols from the input tape and produces output onto the output tape. The output tape becomes covered in symbols, meaning that the transducer is translating from one set to another, or recording the transitions within a set of states from one state to another. This latter example is how the NFST is used in this research. The input to this particular NFST, the one for behaviors, are observations about the state of the field. These might include the position of other players (both on the agent's team and on other teams), the current state of the simulation via state variables, or any other parametric data provided by the simulation to the agents. The output of the system would be the observations from the agent (i.e., another team's agent is observed at a location or transiting between two locations). These observations can come when the agent is standing in a certain state or during any transitions that the agent may make from position to position. The agent movements, if any, are made by the transition function δ. This function takes the input (those observations given to the agent) and crosses them with the set of all states to produce a power set of next states (this is a non-deterministic environment, so any action may lead to several possible states). As a note, if there is no state change, the next state would be the current state (e.g., the agent stands still), and thus the transition would produce an output of ε, resulting in the agent staying in the current state. This ε notation is utilized to

formalize what happens when there is no input from the tape. Rather than producing an error state, or an unknown state, an empty input from the input tape produces a transition from the current state to the current state. In this manner, the output processing and can still occur and the system can continue the simulation (**?**). The transition function, δ, starts from the initial state, position p_0, and continues to process transitions from the current state, s, to the next state, s' (where $s, s' \in S$), until a set of conditions is met ($f \in F \subseteq S$). More detail on the transition function can be found below. The terminating states, f (or accept states in standard NFST terminology), represent those states of completion for the agent (e.g., being eliminated from the game, reaching a boundary in a simulation, etc.).

According to (**?**), the δ^* function can be defined as follows:

- δ^* is the extended transition function. It returns the set of possible states after processing a state and input string pair.
- $\delta^*(q, \epsilon) = \{q\}$ means that if the input is the empty string, the agent will remain at the current state.
- $\delta^*(q, wa) = \cup_{q' \in \delta^*(q,w)} \delta(q', a)$ (i.e., each transition is added step-wise to the history of moves)
- $\cup_{q' \in \delta^*(q,w)}$ is a union over all states reachable on input string w from state q.
- $\delta(q', a)$ is the non-deterministic transition function, returning the set of possible states given state q and input character a

As noted before, the output of the transition function δ is $\mathcal{P}(S)$ (the power set of states reachable from the current state). This power set will provide the simulation framework with the options of moves for the current agent. The system can then use this set of possible moves (those in the power set) to select, probabilistically, the next state for the agent. The set of possible next states will conform to a probability distribution that is either inherent within the strategy (provided by weighting the transitions) or can be learned as the simulation progresses (calculating the weights of the transitions as the simulation progresses).

The NFST takes the set of all possible movements and calculates the next moves for agents based on their particular behavior. Along the way, the NFST will capture observations made by the agents during their transition function δ and produce output in T according to the output function ω. The output function crosses the set of states with the input (which can be empty, as noted previously) and crosses this back to the set of states. The idea of this output is that it is the combination of all possible state transitions given the input. The '*' in ω represents the Kleene star (or Kleene set or closure), or, mathematically, a free monoid. This means that the output represents a set of finite-length strings concatenated from the input alphabet. These strings can be from length 0 to any countably infinite size. In the case of this research, the output is modified slightly. While it does record the state transitions per the normal NFST operation, it additionally outputs the observations of the agents. The result is that the output shows each state in order of transition, but with observations reported in between. To handle this, the NFST utilizes each ε output (i.e., those intervals

where there is no state transition) to write out observations instead of an empty string. All of this output becomes the input to the next stage of the framework, the inference engine (it will take the observations from each agent and aggregate them into probabilistic pathing through the strategy space to determine, via the belief network, the most likely strategy currently in use by the opposing team(s)). This process, taking the output from this transducer and providing it as input to the Strategy Inference Engine, is discussed in Chapter 5 in its own section that documents the Graph Matching algorithm.

The transition function, δ, utilizes a number of factors to determine the next movement from the vector of movements that comprise the core of the behavior. These factors influence the decision on making a movement, the modality of that movement, and the result. In addition, the influencing factors can cause the behavioral chain to be followed backwards (i.e., the agent can progress or regress through the chain when certain influencing factors are met, like the number of active agents is reduced, or the other team's flag has been captured). The general formulation shown here offers insight into how this function operates within the framework, but the exact specification is governed by the models provided to the framework during initial setup. This allows for the system to offer both wide and diverse application of the framework without an inordinate amount of customization.

By way of example, consider the following input to the system. This NFST is given the agent model along with the Total Movement Dependency Diagram (TMDD), of which a particular agent's MDD is a subset. The NFST is also given the set of all

possible states - in this example, they are locations or positions within the simulation space. Given the agent's model, the NFST will iterate through the phases of the model. The behavior NFST is concerned with the movement phase, so this example will focus on that phase but will inform other phases along the way for completeness. The MDD that forms the core of the behavior is provided the current state (part of the input into the system, initially s_0) and then informs the NFST what the next move (or moves in a non-deterministic environment) is (or are) from the current state. The transition function receives these moves and probabilities and processes them. If the probability threshold is not passed, the s', or the next move, is the state output from choosing the movement selected. If the probabilities exceed the threshold for the move probability, then the output is ε, an empty move (i.e., the next state equals the current state, or $s' = s$). As indicated, these moves, either the next state or the empty symbol, are output from the NFST. As a note for completeness, while the agent model is being processed, the agent is also making observations of other agents that it sees on the field or within the simulation environment. These observations are output from the NFST as well and are directed to the Evaluation Engine and the Strategy Inference Engine.

To consider the example practically, some sample data is provided. Figure 3.6 shows the TMDD for one side of a soccer field. This represents all the places an agent could move within this system. Figure 3.7 shows one example MDD for an agent playing defense on the soccer field. This MDD can be extracted, shown in Figure 3.8, and shown as a Behavior model, as shown in Figure 3.9. Placing an

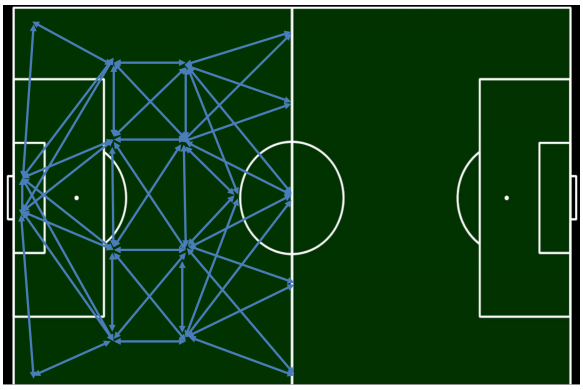


Figure 3.6: Total Movement Dependency Diagram for Soccer

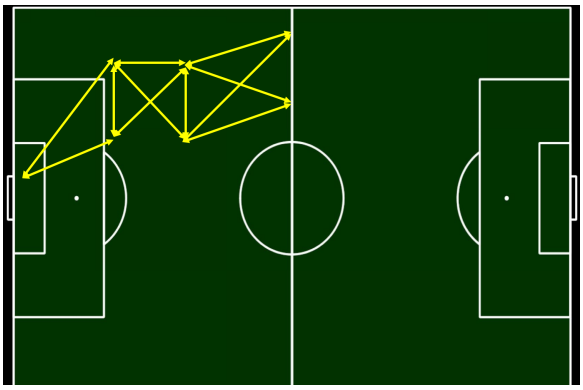


Figure 3.7: Movement Dependency Diagram for a Soccer Agent

agent at the initial state (here s_1, in Figure 3.10), the probable moves become m_0 and m_2 (shown in Figure 3.11). If movement m_0 has a probability of 0.50, and the pseudo-random number generator (PRNG) comes up with a 0.60, then no move occurs using this movement. Similarly, if movement m_2 has a probability of 0.40, and the PRNG comes up with a 0.65, then no move occurs using this movement, either. In this case, an ε is generated as output. The agent remains at s_1 (or, more accurately, the transition model issues a move from s_1 to s_1). The next time the agent model considers a move, the PRNG generates 0.40 and the movement m_0 is taken. In this case, the output is m_0, indicating a transition from s_1 to s_5. As stated before, there are other such calculations being run for each phase of the agent model. The agent continues to move through the phases of this model until the NFST reaches an accepting state, like the end of a period in the soccer match.

The NFST is used to progress agents through their models. These behaviors are assigned to particular agents via a policy, π. The policy, through this mapping shown

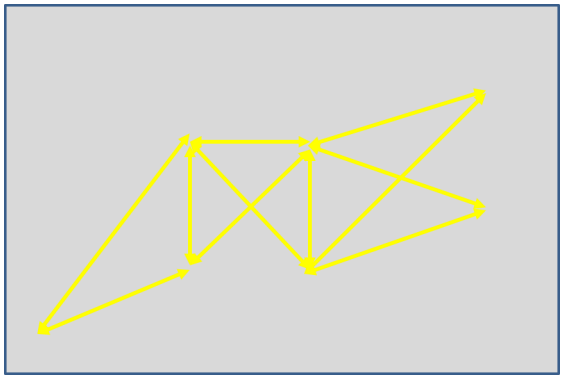


Figure 3.8: Movement Dependency Diagram Extracted

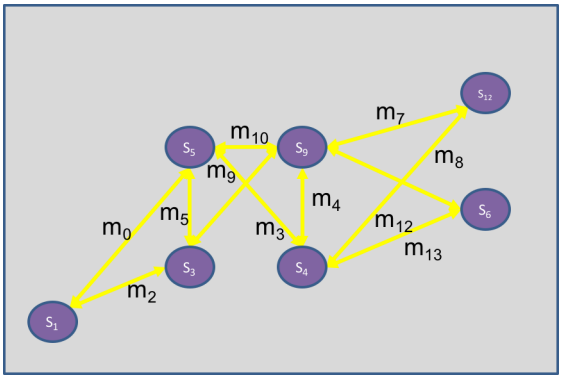


Figure 3.9: Movement Dependency Diagram to Behavior

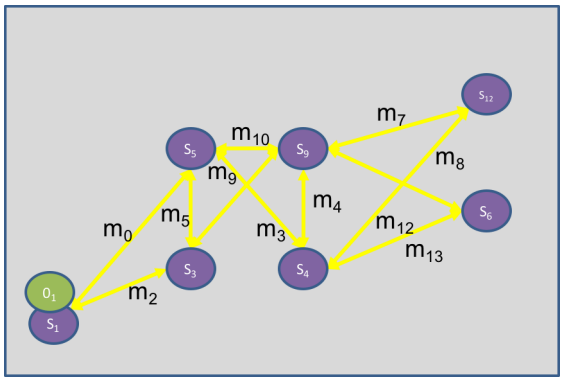


Figure 3.10: Movement Dependency Diagram with Agent

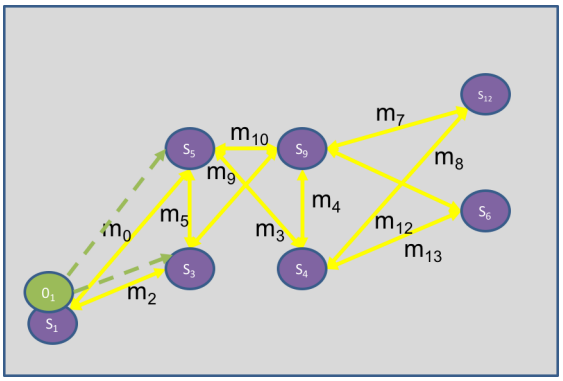


Figure 3.11: Movement Dependency Diagram with Agent Moves

in Equation 3.7, and illustrated in Figure 3.12, assigns a behavior to an agent, and thus imbues the agent with a plan of action (as encapsulated in the behavior). There are also agent control variables with each agent, o, as noted previously in Equation 3.1. These variables may modify decisions made within the behavior, how often such decisions are made, or control elements outside of the behavior (such as enabling group communication). The policy, in assigning the behavior to the agent, provides the control structure for the agent (the control structure is encapsulated within the behavior). Thus, changing policies for an agent gives the agent a whole new behavior and changes the way that agent works within and interacts with the world.

$$\pi : o \rightarrow b \tag{3.7}$$

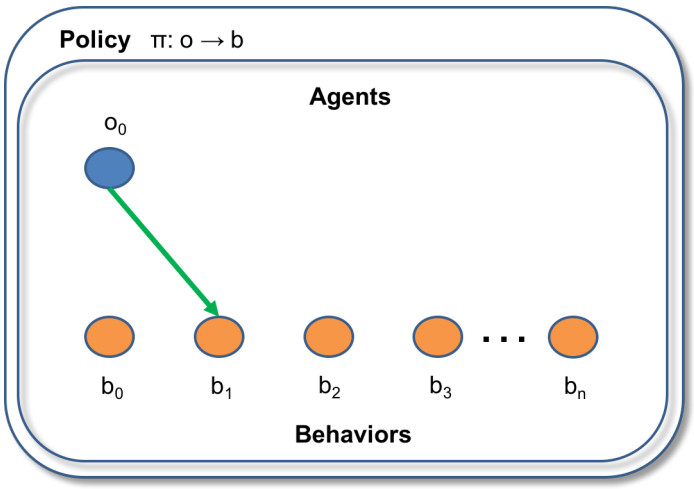


Figure 3.12: Policy Mapping

A strategy, σ, is the mapping of all agents, O, within a team, τ, to their selected policies (each strategy has access to Π_s, a subset of all policies, as in Equation 3.9).

Because this is a set to set mapping (i.e., the set of agents on a team to the subset of policies available to the team) the mapping is described as a cross-product of $\tau \times \Pi_s$. This cross-product results in a policy being assigned to each player on the team, π_{o_i}. This strategy can be thought of as a team policy. A team, τ, is a set of agents, as shown in Equation 3.8. In a perfect world, this matching of agents to behaviors would be done optimally - every agent with the perfect role and assigned the perfect behavior. If this is not known *a priori* then it is vital that the strategy in place be switched until a 'best fit' is achieved (measured by overall team performance as measured by the Intelligent Strategy Selection Engine, covered later). If the optimal matching is known ahead of time, this matching can be done initially. The mapping is shown in Equation 3.10, and is illustrated in Figure 3.13.

$$\tau = \{o_0, o_1, \ldots o_n\} \tag{3.8}$$

$$\Pi_s = \{\pi_0, \pi_1, \ldots \pi_j\}, \Pi_s \in \Pi \tag{3.9}$$

$$\sigma : \tau \times \Pi_s \rightarrow \pi_{o_i}, \forall o \in \tau \tag{3.10}$$

With the strategy defined, a strategy-based system can evaluate the current policy in force (i.e., the current policy being utilized by a particular agent) for each agent according to the valuation function and compare it with the expected value of other

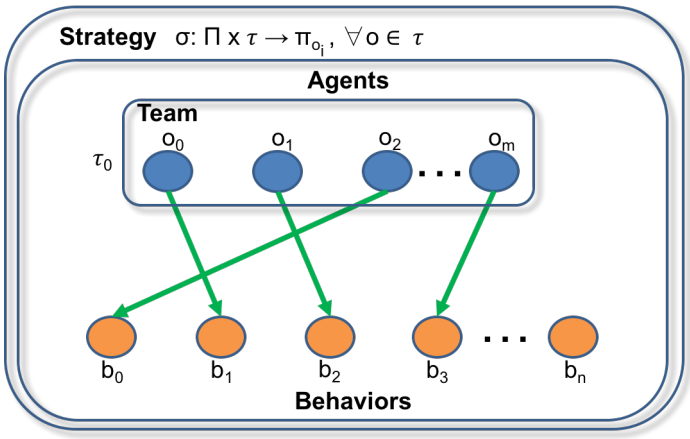


Figure 3.13: Strategy Mapping

possible policies. The comparison between any two policies is the difference in their valuations. When the difference is great, making a change to the better policy is clear. However, when the difference is minimal then additional consideration should be used. Since there is a cost to switching policies (because of the reassignment of behaviors to the members of a team due to their policy shifts), there should only be policy change when the difference in the two policies' valuations exceeds a threshold variable. When there is a threshold reached (meaning that the candidate policy's performance is more than ϵ greater than the current policy) the switch to another policy is initiated. Here, ϵ is the threshold variable designed to keep policy vacillation to a minimum (lest the Strategy Inference Engine spend all of its time switching back and forth between two similarly performing policies). This process is detailed in Chapter 4.

Each strategy is a mapping of policies that reflect a certain motif, plan of action, posture of readiness, or any such concept that directs agents at the highest level

towards a larger goal. For example, adopting several defensive policies may create a strong defensive strategic posture, but adopting a mix of policies may create a balanced strategic posture. The formation of the policies and strategies can happen either through applicable domain knowledge (known behaviors, tactics, policies, and strategies) or through machine learning. In the case of domain knowledge the system is utilizing encoded models provided by experts in the field. When machine learning is used, the system utilizes a default model. This default model moves the agents probabilistically as they learn and then modifies its model based on performance. In the experiments found in the Experiments Chapter (Chapter 7), the former was used - domain knowledge was gathered from actual participants, coaches, and literature on the event / sport. The strategies were configured similarly by aggregating complementary policies according to experts in the field. It should be noted that these subject matter experts had no concern for the simulation's ability to reason, learn, or infer; fidelity to reality was of the utmost concern and the simulation accomplishes this. In the absence of such domain knowledge, the framework can be utilized to gather this information through usage. As the system collects any new strategies that it acquires along the way, so the more it encounters, the more the knowledge base grows.

These strategies are then able to be acted upon by the the Intelligent Strategy Selection Engine (ISSE). The ISSE can ensure that the best strategy is in place for each team. It takes in strategy inference data and produces the next strategy that should be in place. The ISSE can then be represented by another non-deterministic

finite state transducer, defined as before with behaviors. Recall that an NFST is defined generically as a tuple $(\Sigma, \Gamma, S, s_0, \delta, \omega, F)$, where Σ represents the input alphabet, Γ represents the output alphabet, S is the set of all possible states, s_0 is the initial state, δ is the transition function, ω is the output function, and F is the set of final states (accepting states).

As applied to this work, the ISSE representation is a modification of this generic formulation to utilize the variables of the system. This is shown in Equation 3.11 and illustrated in Figure 3.14.

Figure 3.14: Intelligent Strategy Selection Engine

$$ISSE =< A, \Gamma, \Sigma, \sigma_0, \delta, \omega, F > \tag{3.11}$$

where

- A is the input set (set of output from the SIE and the EE)
- Γ is the output set (set of all simulation outcomes)
- Σ is the set of all states (set of all strategies)
- σ_0 is the initial strategy
- δ is the transition function ($\delta : \Sigma \times (A \cup \{\varepsilon\}) \rightarrow \mathcal{P}(\Sigma)$)
- ω is the output function ($\omega : \Sigma \times (A \cup \{\varepsilon\}) \times \Sigma \rightarrow \Gamma^*$))
- F is the set of conditions of completion for the simulation

The transition function, δ, is a multi-variate calculation that takes in the input to the NFST (i.e., the output from the SIE and the EE) and crosses it with the set

of all available strategies to produce the next strategy that the team should put in force (N.B., this may be the current strategy, in which case an ε is the input and the strategy remains unchanged). This transition function decides if the current strategy is the best option through its valuation function (an aggregate performance score for each of the strategies). If the current strategy is not the best performer according to this metric, the transition function returns the best of the available strategies and puts that strategy in force for that particular team. The valuation may be based on the result of the score of the game, a battlefield assessment, the ownership of key environmental elements or positions, or any of many such conditions within the simulation.

The output function, ω, is producing both a record of state transitions (when a new strategy is selected for a team) and statistical data on the progress of the simulation. As before, when there is no shift of strategies for a given team the simulation can use the ε output to produce an object instead. In this manner, statistical information can be encapsulated into the objects and included in the output. The data in this object includes information about the individual agents, aggregate team data, environmental transitions, and historical data about the progression of the simulation (including the various behaviors, policies, and strategies in place along with their relevant data).

The diagram in Figure 3.14 shows the overall design of the ISSE. For each team, τ, that it is evaluating it has a subset of strategies, Σ_s. Each strategy, σ, is a mapping of a policy, π, to to each member of a team, o. This is illustrated in the figure where each of the strategies and their related policies are shown.

3.7 Summary

This strategy-based specification can be developed into models that encapsulate the information that describe strategies. This information, encoded into the models, contains sub-models, performance variables, and other data to determine how the strategies are implemented within the system. The framework is designed to be open ended. It can be easily customized to many different types of simulations and scenarios without having to manipulate the inner workings of the framework. To create a new scenario the user creates the underlying models, defines the arena that contains the simulations, and provides the running data (number of players, policies, strategies, etc.) and feeds these into the simulation engine. The SiMAMT framework, detailed in Chapter 4, can then utilize this new data to create, run, and evaluate multi-agent multi-team systems based on this new configuration. One such implementation is described in detail in the Chapter 7.

The following experiments demonstrate the concepts of strategy, strategy-based systems, and offer insight into their performance.

3.8 Experiment 1: Strategic Interaction in Roshambo and RPSLS

3.8.1 Introduction

The concept of strategy is important to this research, and along its development several experiments were conducted. This section offers one such example of

implementing strategy into real-world systems. The conclusions are not groundbreaking, in fact, they are the expected outcomes, but this is a verification that the proposed strategy-based framework performs exactly as expected. The experiment provides a strategy-based view of the popular game of Roshambo (later expanded into RPSLS). While this game is simple, it is designed to show a minimally sufficient case for strategy-based system and provide an initial proof that strategy matters. Additionally, it shows that the implementation, modeling, and simulation of strategy is viable.

3.8.2 Specification

This experiment lays the foundation for understanding the processing and selection of strategies. By way of example, consider the game of Roshambo. In this game the strategy, σ, can be viewed as a mapping of a subset of policies, Π_s (drawn from the set of all policies Π), to an agent. The strategy-based system takes into consideration weighted views of previous actions of the focus agent, the actions of the other agents within view (meaning observable), and the predicted future actions of those other agents. In Roshambo each player can choose to play rock (a_r), paper (a_p), or scissors (a_s). The other player can also choose from the same set of choices. Because this is a single-agent system, the policy and the behavior are synonymous (i.e., the policy holds the plan of action for the agent just like a behavior normally would), so the term policy will be used as if it were a behavior (recall that normally a policy maps an agent to a behavior, but here it simply substitutes for it in the single-agent system).

For the isolated view, where only the moves of the player are used, a strategy σ can be viewed as a mapping of the set of policies that select and weight the combination of the history of moves, the current probabilities of the next move, and the prediction of future moves. Equation 3.12 (as presented in the preceding section) shows how a strategy-based system can use σ (Equation 3.13) to select the best policy (denoted as π^* (of the format shown in Equation 3.14, a tuple) from its available list of policies (the subset of policies, Π_s ($\Pi_s \in \Pi$), in the strategy σ) by evaluating each and choosing the best policy, π^*, that produces the maximum value (as determined by Equation 3.15). Thus π^* should lead to the optimal choice of the next action, a' (Equation 3.16) to take (in this case, the best choice of rock, paper, or scissors) as determined by the maximally valued action a from Equation 3.15. In the case that the current policy in force, π, is not producing optimal results, the strategy-based system can select a new policy from the strategy σ to implement (π', the next policy) via Equation 3.13. For the policy, Equation 3.14, LB represents the number of steps to look back, α_{LB} is the learning rate for looking back, w_{LB_n} is the weight of the look-back moves (where $n = \{r, p, s\}$), LA the number of steps to look ahead, α_{LA} is the learning rate for looking ahead, w_{LA_n} is the weight of looking ahead, w_r, w_p, and w_s are the weights for each move, E_r, E_p, E_s are the expected values for making each move. The valuation function, shown in Equation 3.15, is used to evaluate each policy to determine its performance. Additionally, it determines the best next action (a') via the max function. In this equation, the valuation for each policy, V^{π_n}, is summed from three elements evaluated across each action (i.e., $\max_a$ finds the action, a, that

returns the maximum value). The first element, $\left(\sum_{k=0}^{LB} w_{LB_k} r_{a_{t-k}}\right)$, is the total for the 'look-back' moves. Where k is the number of steps to look backwards, sum up the weight of each previous turn multiplied by the reward, r, of each previous turn. The previous action is determined by a_{t-k}, or the action taken at the current time minus the current look-back value. The second element, $(\max_a(w_a r_a))$, is the value of the highest valued move for the current action (i.e., a_t is the action at the current time). The third element, $\left(\sum_{l=0}^{LA} w_{LA_l} E_{a_{t+l}})\right)$, is the total for the 'look-ahead' moves. Where l is the number of steps to look ahead, sum up the weight of the next turn multiplied by the expected value of taking that action. Once these are summed, the value for each a is compared with the all other actions and the maximum a is determined (this is the implication of the $\max_a$ operator).

$$ISSE =< A, \Gamma, \Sigma, \sigma_0, \delta, \omega, F > \tag{3.12}$$

$$\sigma : \tau \times \Pi_s \rightarrow \pi, \text{subject to } \underset{\pi}{\operatorname{argmax}}\, V^{\pi_n}(s_t), \forall \pi \in \Pi_s \in \sigma \tag{3.13}$$

$$\pi =< LB, \alpha_{LB}, w_{LB_r}, w_{LB_p}, w_{LB_s}, LA, \alpha_{LA}, w_{LA_r}, w_{LA_p}, w_{LA_s}, w_r, w_p, w_s, E_r, E_p, E_s > \tag{3.14}$$

$$V^{\pi_n} = \max_a \left[\left(\sum_{k=0}^{LB} w_{LB_k} r_{a_{t-k}} \right) + \left(\max_a (w_a E_a) \right) + \left(\sum_{l=0}^{LA} w_{LA_l} E_{a_{t+l}}) \right) \right] \tag{3.15}$$

$$V^{\pi_n} \rightarrow a' \tag{3.16}$$

The procedure for determining the next move, a', is presented next. The first elements calculated in the move determination phase are the individual likelihoods, L_n, where n is the particular move rock, paper, or scissors (i.e., L_r is the likelihood for a_r, the action choosing 'rock'). These likelihoods are the weighted and discounted values of choosing k by looking backwards LB moves and predicting forward LA moves (Equations 3.17,3.18,3.19). In these formulae the individual likelihoods, L_n, are cumulative values of the rewards received from the past moves added to the cumulative predicted rewards of taking the future moves. As indicated, these individual values are weighted according to how far back or how far forward the formula is looking and weighted accordingly as well. A particular policy π gives the number of steps to look back and the weighting of those values as well as the number of steps to look forward and the weighting of those values. Thus the policy customizes these formulae to suit the strategy σ by adjusting these variables (look behind, weights, learning rates, etc.).

$$L_r = \left[\left(\sum_{k=1}^{LB} \frac{1}{k} \alpha_{LB} w_{LB_r} r_{a_{t-k}} \right) + \left(\sum_{k=1}^{LA} \frac{1}{k} \alpha_{LA} w_{LA_r} E(a_{t+k}) \right) \right] \tag{3.17}$$

$$L_p = \left[\left(\sum_{k=1}^{LB} \frac{1}{k} \alpha_{LB} w_{LB_p} r_{a_{t-k}} \right) + \left(\sum_{k=1}^{LA} \frac{1}{k} \alpha_{LA} w_{LA_p} E(a_{t+k}) \right) \right] \tag{3.18}$$

$$L_s = \left[\left(\sum_{k=1}^{LB} \frac{1}{k} \alpha_{LB} w_{LB_s} r_{a_{t-k}} \right) + \left(\sum_{k=1}^{LA} \frac{1}{k} \alpha_{LA} w_{LA_s} E(a_{t+k}) \right) \right] \tag{3.19}$$

These totals, L_r, L_p, L_s, can then be combined to determine the proportionate probability of choosing each move by considering each with respect to the total of all moves (Equations 3.20,3.21,3.22).

$$P_r = \frac{L_r}{(L_r + L_P + L_s)} \tag{3.20}$$

$$P_p = \frac{L_p}{(L_r + L_P + L_s)} \tag{3.21}$$

$$P_s = \frac{L_s}{(L_r + L_P + L_s)} \tag{3.22}$$

The next action (or move, in this case), a', can then be calculated by determining the maximum value from the inverse proportions weighted across each move (Equation 3.23). The rationale behind the inverse proportions is that the goal is to choose according to the least likely moves (inverting the most likely moves).

$$a' = \max_a((1 - P_r)w_r, (1 - P_p)w_p, (1 - P_s)w_s) \tag{3.23}$$

In this way, the aggregate probabilities of each move, based on the past selections of that same move and the future options for that same move, have combined to form probabilities for each move. The next best move can then be selected by these probabilities. A learning rate, α, is added to determine the overall influence of these former and future moves. The learning rate, α, is a common variable used in machine learning and artificial intelligence to decide how much of what is learned at each step is used in the calculation. It is often a small number so that no one move overwhelms the experience the agent has gained over the last several iterations.

While this formulation, Equations 3.17 - 3.23, allows a primitive strategy based on how often the player should repeat moves, which moves it should favor, and how to interpret its experience over time, it does not consider the moves of other players. It does, however, lead into the intuition behind such considerations. Before moving away from this strategy representation under isolation it should be noted that there is more packed in to it than may at first be obvious. First, anytime reward (the value given for attaining a certain state) is considered for future moves under uncertainty (i.e., in the absence of a model or an approximate model formed through experience) it can be replaced with an expectation of reward, as shown in Reinforcement Learning (RL) (Russell and Norvig, 2003). This expectation of reward uses probabilities and likelihoods to determine most-likely rewards under non-determinism. Second, the resultant move calculation (Equation 3.23) could produce an ordered vector of moves with their probabilities (ordered by the valuation, Equation 3.15) . This ordered vector would rank all moves, above some threshold ϵ, so resolving the best choice

would be looking up the highest indexed move. This ϵ provides the ability to have a default override in the cases where there is no clear cut 'winner' for the best move. The threshold value is a minimum floor for move calculations, so only those moves exceeding the threshold would be considered. This enhances the strength of the policy as it allows for probabilistic drive when the move choices are high in value, but an underlying set of choices to rule the behavior when the move choices are lower valued. Third, this ordered vector could also produce a transition matrix to govern the probabilities in situations where states were used (instead of just actions) to produce the best a' (as shown in Equation 3.24).

$$P = \begin{bmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{bmatrix} \tag{3.24}$$

$$x_0 = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{3.25}$$

$$x_1 = x_0 P = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{bmatrix} = \begin{bmatrix} 0.9 & 0.1 \end{bmatrix} \tag{3.26}$$

$$x_2 = x_1 P = x_0 P^2 = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{bmatrix}^2 = \begin{bmatrix} 0.86 & 0.14 \end{bmatrix} \tag{3.27}$$

As an example, taking the transition probability matrix P (Equation 3.24) and the sample initial matrix x_0 (Equation 3.25) and the learned matrices x_1 and x_2

(Equations 3.26, 3.27), the resultant transition matrix is also shown in (Equation 3.27). This shows that the transitions can be formed conditionally based on the initial transition probabilities (i.e., the current policy in-force according to the underlying strategy) and first transition (i.e., the previous choice or choices). This can continue for as many choices as necessary, as shown in (Equation 3.28) and resulting in (Equation 3.29).

$$x_n = x^{n-1}P \tag{3.28}$$

$$x_n = x_0P^n \tag{3.29}$$

The strategy can also, for the purpose of a move based game like Roshambo, be represented in policies that contain the weights of each move and the α for each. The weights could be proportioned to favor irregularity (avoid repetition), favor regularity (repeat favorite moves), or follow pre-determined patterns. Each of these policies would be contained within the set of policies collected in the strategy. In addition, instead of using the same α for each move, this could be customized per move, further enforcing the behavior specified by the current policy in force selected by the strategy. Alternatively, the formulation for the influence of α can shift from Equation 3.30 to Equation 3.31 (presented here with n to represent any move), moderating α by the step size k. This would reduce the influence of each calculated value on the total calculation in proportion to how far back or how far forward the value is.

$$L_n = \left[\left(\sum_{k=1}^{LB} \frac{1}{k} \alpha_{LB} w_{LB_n} r_{a_{t-k}} \right) + \left(\sum_{k=1}^{LA} \frac{1}{k} \alpha_{LA} w_{LA_n} E(a_{t+k}) \right) \right] \tag{3.30}$$

$$L_n = \left[\left(\sum_{k=1}^{LB} (1 - \alpha_{LB})^k w_{LB_n} r_{a_{t-k}} \right) + \left(\sum_{k=1}^{LA} (1 - \alpha_{LA})^k w_{LA_n} E(a_{t+k}) \right) \right] \tag{3.31}$$

In the experiments for strategic modeling, the Roshambo models were developed. As indicated in Equation 3.14, the strategy σ needs to contain a subset of policies ($\Pi_s \in \sigma \in \Pi$), each of which is a tuple of the various features. Expanded out, the tuple contains the look back steps, alpha for look back steps, weight for looking back at rock, weight for looking back at paper, weight for looking back at scissors, look ahead steps, alpha for looking ahead, weight for looking ahead rock, weight for looking ahead paper, weight for looking ahead scissors, expected reward for rock, expected reward for paper, and expected reward for scissors. An example tuple, for a policy called 'LB3Rock' (for Look Back three steps with a rock bias), is 3, 0.2, 0.67, 0.16, 0.17, 0, 0, 0, 0, 0, 0.67, 0.16, .0.17, 1, 1, 1. This policy π would look back at the previous three moves made and weight them with a bias to rock, then consider the current move choices with a rock bias, but there is no way to consider future moves. Additionally, in this iteration of the game there are equal rewards for each move. The two player game was run with two independent policies (as these are run in isolation) and compared. These experiments and their results are found in the next section.

Policies, for which plenty of foundational work has been done in Machine Learning (Russell and Norvig, 2003), are well established. The addition of a higher level strategy is put in place to help select from among these policies. Suppose, by the example, that the 'LB3Rock' policy, π_{LB3R}, was performing poorly. The ISSE may evaluate the strategy in place, σ_1, and, based on its performance, may select another policy from its subset of policies, say 'LB3Paper', π_{LB3P}. In this manner, the ISSE is using strategy σ_1 by monitoring performance and making a decision about which of its policies should be in place. The more data and experience the ISSE has with the various policies within the strategy the better this selection will be. As history is accumulated, and the strategies of the other player are considered, the selection of the best policy π^* from the set can happen more quickly and accurately.

The experimental results, under Experiments below, verify the concept and utility of strategy as a model for agent interaction (here, a single agent versus another single agent, but both agent's actions must be considered for every move).

3.8.3 Proving the Viability of Strategic Modeling

To prove the viability of strategic modeling, the Roshambo and RPSLS experiments were implemented. The purpose of these experiments was to establish some baseline reference for the playing of the game for comparison, implement policies within the game to measure their impact, and then create and test strategies that utilize these policies. Once the baseline performance was established, the goal was to have the strategies square off against each other to see if it is possible to determine which

policy the opponent has in place (and thus infer their strategy), if and when switching policies should occur, and if shifting strategies is possible. This user study was approved by an Institutional Review Board (IRB). Subjects were asked to play Roshambo and record their moves each turn. These records became the real-world baseline for measuring the quality of the simulation. In each case, the human subjects proportions of moves, distribution of moves, and overall win / loss ratios matched the simulation. This established that the simulation of Roshambo and RPSLS was valid *w.r.t.* fidelity to the real-world models. Additionally, the lists of moves from the subjects was analyzed to determine their move bias which informed the various policies in the simulation.

A simple model, here implemented as a Finite State Automaton (FSA), for Roshambo is shown in Figure 3.15, along with the same model as defined by implementing the policy in *RockBias* strategy, shown in Figure 3.16.

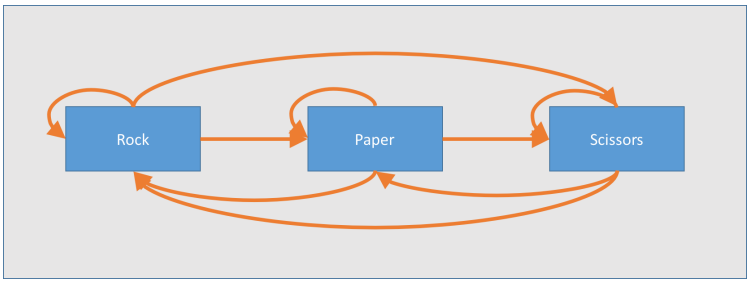


Figure 3.15: Roshambo FSA Model

These games were played with each of the two players implementing a strategy from the list, Table 3.1. Each game consists of 10,000 turns where each player chooses rock, paper, or scissors depending on their strategy and policy that is currently in place. The winner is determined and recorded. The possible outcomes are win, lose,

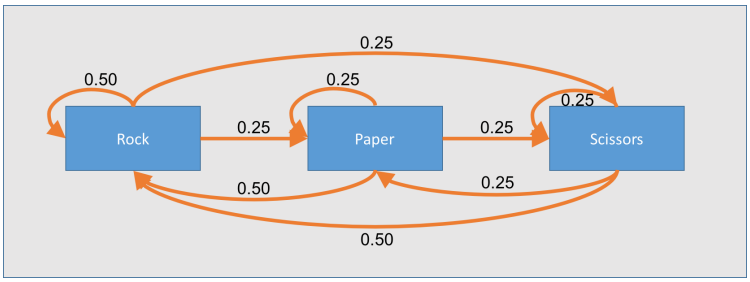


Figure 3.16: Roshambo FSA Model - Rock Bias

Normal	Normal, LookBack3, LookBack10
RockPref	RockBias, LB3Rock, LB10Rock, LB10RockEven
PaperPref	PaperBias, LB3Paper, LB10Paper, LB10PaperEven
ScissorPref	ScissorsBias, LB3Scissors, LB10Scissors, LB10ScissorsEven
LizardPref	LizardBias, LB3Lizard, LB10Lizard
SpockPref	SpockBias, LB3Spock, LB10Spock

Table 3.1: Strategies and related Policies for Roshambo and RPSLS

or draw. The list of policies is shown in Table 3.2. In this table, the Roshambo policies are listed without an asterisk, and the RPSLS are preceded by an asterisk. In each experiment, the strategies were tested against each other in sets (e.g., RockPref, ScissorPref) with the base for comparison being Normal vs. Normal (see Table 3.2 for explanations).

3.8.4 Results

The end result of these experiments are shown in Table 3.3. The normal vs. normal games showed the expected distribution where any one player, on average, wins about 33% of the time. It is also noteworthy to consider the not-lose percentage, that is the percentage of time that there is not a negative outcome. For each of these experiments with normal vs. normal the not-lose percentage was around 67%. This falls along

Normal	Random selection of R, P, or S
RockBias	Random selection, but Rock is three times more likely
PaperBias	Same, but for paper
ScissorsBias	Same, but for scissors
*LizardBias	Same, but for lizard
*SpockBias	Same, but for Spock
LookBack3	Random selection of R, P, S, looking back three moves
LB3Rock	Rock bias with three move look back
LB3Paper	Same, but for paper
LB3Scissors	Same, but for scissors
*LB3Lizard	Same, but for Lizard
*LB3Spock	Same, but for Spock
LookBack10	Random selection of R, P, S, looking back ten moves
LB10Rock	Same, but with looking back 10 steps
LB10Paper	Same, but for paper
LB10Scissors	Same, but for scissors
*LB10Lizard	Same, but for Lizard
*LB10Spock	Same, but for Spock
LB10RockEven	Look back 10, but with no bias on lookback for rock
LB10PaperEven	Same, but for paper
LB10ScissorsEven	Same, but for scissors

Table 3.2: Policies for Roshambo and RPSLS (* precedes RPSLS Policies)

the expected lines of a standard probability distribution. The research objective was to determine if the strategies being modeled (which are meant to more closely resemble real-world player policies and patterns) will provide the advantage that they should. In the experiment of LB3RockBias vs. LB3PaperBias, the paper-bias won 43.78% with a not-lose percentage of 71.74%. Other match-ups show similar results where the superior strategy wins more and loses less. These comparisons reveal that mismatched strategies show an improvement over random strategies. They also show that the strategy models are functional. The intuition is that these experiments are challenged by the fact that there are only three choices, so any random guessing will succeed about 33% of the time.

3.8.5 Expanding the Complexity of the System

To test this intuition the number of choices was increased from 3 to 5, shifting the game from Roshambo to Rock, Paper, Scissor, Lizard, Spock (RPSLS). This game has the same rules, but adds two new options. In this version of the game, as shown in Figure 3.18, there are two ways to win and two ways to lose (and, of course, one way to tie) with each choice. With two ways to win, two ways to lose, and one way to tie, the results of random guessing should be 40% wins, 40% losses, and 20% ties based on standard probability. The results, shown in Table 3.3, show that the experiments supported the probability hypothesis with the wins for the normal play being 40.13% with a not-lose of 60.22%. With this in mind, strategy was implemented with the expanded list, shown in Table 3.2. In the list of policies, Table 3.2, those policies

specific to Lizard and Spock are marked with an *. This game was fully implemented with the same strategy representations. One such strategy match-up, LB3RockBias vs. LB3PaperBias, showed a significant improvement in win percentages (from 43.78% with only three choices (RPS) to 62.43% with the five choices(RPSLS)) and in not-lose percentages (from 71.74% to 87.47%). This shows that having more choices allows for strategy to be more fully implemented and results in more significant differences in strategy and non-strategy (also shown in Table 3.3). This is further evidence of the veracity of the strategy model presented herein.

With the game expanded as explained above into RPSLS, the SiMAMT system works in exactly the same manner, with only the change in the model. The system will now simulate the new game by only switching out the previous model with the new model, shown in Figure 3.17, but without any other changes to the system. This model is color-coded so that it is easier to follow the flow from one state to another. Each state and its outbound transitions are the same color.

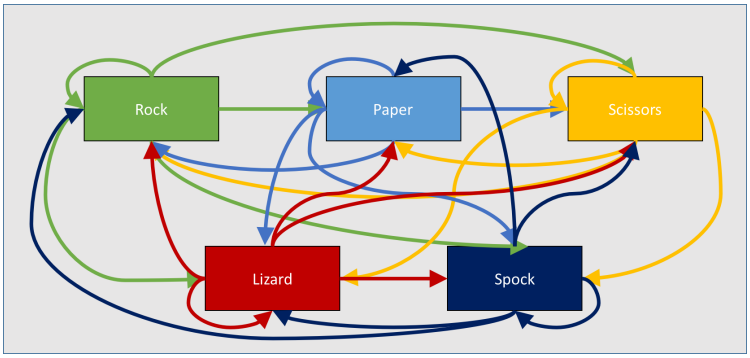


Figure 3.17: RPSLS FSA Model

These results also lead to an area of future research. The future work question is this: is there a minimum size or a minimum number of choices required for strategic

Game	Bias	Player1 Win	Player1 Not Lose
RPS	Normal vs. Normal	33.29%	66.33%
RPS	Paper vs. Rock	43.78%	71.74%
RPSLS	Normal vs. Normal	40.13%	60.22%
RPSLS	Paper vs. Rock	62.43%	87.47%

Table 3.3: Results of RPS and RPSLS with Strategies

reasoning to be effective? This questions asks if there is a minimum complexity threshold for strategic interactions, and this will be addressed in future work. The evidence suggests that the influence of strategy use is more prevalent as complexity increases (i.e., those agents following a strategy routinely outperform agents following a base policy, like random guessing). In this experiment, the number of choices was indicative of such a threshold.

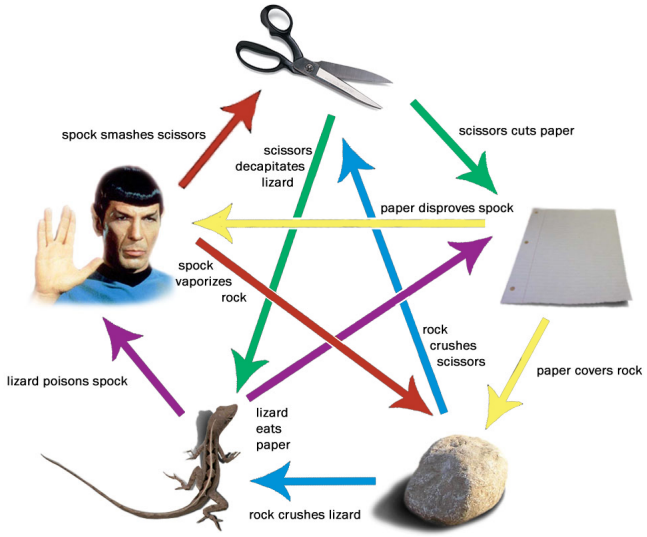


Figure 3.18: Gameplay Diagram of Rock, Paper, Scissor, Lizard, Spock (MU, 2012)

3.8.6 Conclusions

This experiment shows that strategy can be modeled, implemented, and simulated. Further, it is clear, even from these simple results, that strategy offers an advantage in real-world scenarios where strategic actions matter.

3.9 Experiment 2: Multi-Agent AI using Strategy-Based Reasoning

3.9.1 Introduction

This second experiment moves the strategy-based reasoning to a simple video game. This experiment progresses the study from the single-agent Roshambo and RPSLS experiment to a multi-agent example.

Classical artificial intelligence adds a depth to gameplay that mimics playing against another human. This desirable trait offers gamers a richer, more meaningful experience. While the ultimate AI might be perfectly analogous to a human opponent, the reality often falls short. There is a delicate balance that must be achieved between an invulnerable omniscient intelligent agent and a rudimentary walking target. This balance is challenging to create and even more difficult to maintain. Game creators are tasked with using the state of the world as the game understands it (which describes every object within it in perfect all-knowing detail) to create a character to mimic human behavior (which does not have such complete knowledge). As a result, the AI is often so simplistic that it becomes nothing more than walking background set

decoration in an effort to keep it vulnerable. Also, because this type of character creation is difficult, programmers often take shortcuts. There are many examples available where the game presents the player with several simple AI characters in an effort to mask their vulnerability. This initially seems to offer the presentation of greater difficulty but this charade quickly falters. It is presumed that this team of enemies will work as a cohesive unit and take advantage of their greater numbers, that they will reason and act in a strategic way, and that their advantage is insurmountable. The usual reality, however, is that this is not true. In most cases this team of AI characters is really just multiple instances of a single character AI. Due to the fact that this AI is not exhibiting teamwork, there is often a 'trick' (an unintended action or set of actions that circumvent the challenge that was intended by the game designer) that undermines the true nature of this weaker AI.

Learning in multi-agent systems is challenging. There is a lot of detail to be considered in the arena of teamwork in games. There is much to be learned from the research in multi-agent systems. This research informs our efforts in this arena. We wish to design an engaging AI that will take advantage of outnumbering the player to reason and act as a team. By acting as a team we wish to show that even a simple AI can create a stronger challenge than the simple, single-agent method. In so doing, we will create an AI that feels more natural (e.g., acts like a human-controlled player) and avoids these tricks that allow it to be circumvented. By applying lessons learned from multi-agent system research we can have the agents themselves reason in an informed manner with the knowledge that they are part

of a team. While multi-agent research does not currently go so far as to begin to form strategic reasoning, we wish to show that it can. Strategic reasoning arises when the team of intelligent agents uses the available information in a way that exhibits a collective intelligence. Additionally, the agents' actions are given from the decision of the collective intelligence rather than from the agent individually. This is the contribution of this experiment, to reason collectively (i.e., work together as a team rather than as multiple individuals) and demonstrate, even in such a simplistic environment, that this reasoning is advantageous.

We can either assume that these agents can communicate their intentions to each other or presume that there is only certain knowledge that is available to each. In either scenario we can show that the resulting performance is better than the multiple single-agent schemes.

While this experiment is centered in a game arena, these findings have important ramifications in other areas as well. First, we do hope to improve the AI in games by making these teams of reasonable intelligent agents perform in such a manner that the aggregate challenge they present matches the goals of the game designers. Second, moving beyond the casual game arena and into the serious games and simulations arena, we can see that this type of AI could be used to model critical systems and interactive environments. Finally, we can see that multi-agent systems can benefit from this application of collective intelligence.

3.9.2 Implementing Multi-Agent Teams

Our goal is to show that multiple agents operating as a team (that is, sharing information and goals) will outperform multiple agents implementing single-agent artificial intelligence. To do this we devised a simple game that pits one player against three opposing agents. The paradigm that we are using is a pursuit game with no obstacles. The test subject controlled player's initial position is on the left side of the screen in the middle. The three AI-controlled opponents are placed on the right side spread out evenly (Figure 3.19). Once the game begins, the opponents pursue the player until caught. The game is configured to vary the speed of both the player and the opponents and to vary the strategy that the opponents are using (e.g., direct-pursuit, aim-and-miss).

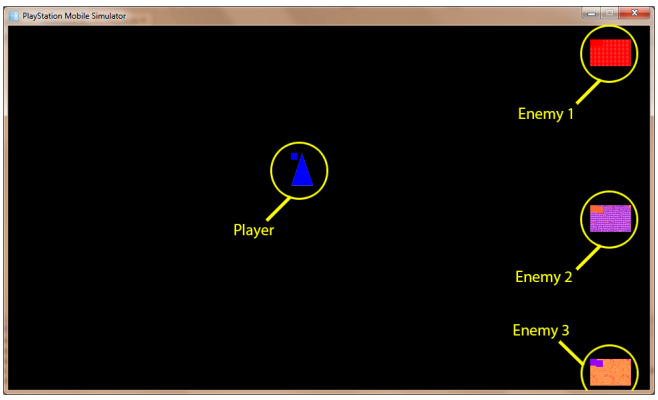


Figure 3.19: Initial Starting Positions

Each team (the single player and the multiple opponents) can be assigned a strategy from the ISSE. For the single player, they can be assigned a strategy (then the game is a simulation without human control) or they can be controlled by a

human player. For the three-agent team, the strategy assigns a policy to each member of the team. For this research we used two different strategies, each of which contains policies. The first strategy used was direct-pursuit (DP). In this strategy each agent is assigned a policy where they directly pursue the player. Even though there are multiple agents in the game, they are all operating independently. The second strategy used is called aim-and-miss (AAM). This strategy seeks to plot the direct line to the player and then coordinate that information with the same information from the other opponents. This direct line pursuit angle is then adjusted based on the current formation. A formation is a behavior and is the arrangement of the opposing agents with respect to the player. While several are possible, and many were considered, this current research just explored one formation to isolate the strategy-based artificial intelligence component.

The model utilized by the agents is a Finite State Automaton (FSA), shown in Figure 3.20. This model, while simple, conveys the actions of the autonomous agents within the system. The simulation moves the agents through each phase of their operation. In a continuous loop, the agent senses its surroundings (namely, the location of the other agents and the location of the target), it shares this information with the other agents on the team, and moves. The move model phase can be shown with its sub-model, Figure 3.21. Here, the move phase is broken down into sub-phases, depending on the sensing. In each scenario, the agents have roles assigned to them (part of their behavior). If the agent is the lead agent, then the weight of the direct phase is highest. If the agent is to the right of the lead agent, the miss right

weighting is highest, and if the agent is tot he left of the lead agent, the miss left weighting is dominant. The weights of the other components are not zero because the location of the target and the other agents may necessitate a move that would be better served by another phase of the move sub-model. In any case, the sub-model returns the move that the agent should make based on the sensing and sharing data gathered in the other phases and the move calculation performed by the sub-model. For different behavior models, the system would work identically, but the models would be different. This is a highlight of the the SiMAMT system, that with a simple change in models the system can simulate different agents, strategies, behaviors, etc., without having to rework the framework itself.

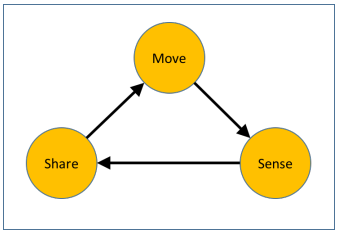


Figure 3.20: Aim-and-Miss Agent FSA Model

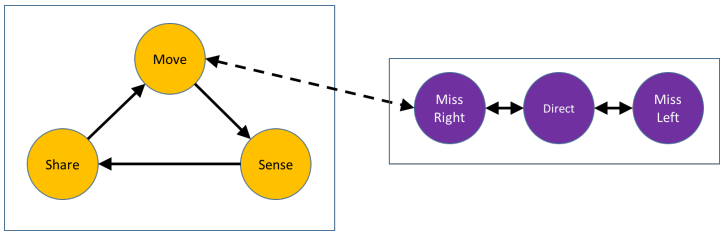


Figure 3.21: Aim-and-Miss Agent FSA w/ Move Sub-Model

The behavior chosen was the C-formation where the center opponent performs direct-pursuit on the player and the other two opponents form a reverse C to close on the player like a claw (Figure 3.22, Figure 3.23). This means that, in this case,

the strategy assigns the pursuit policy to the central agent on the team and assigns the aim-and-miss policy to the other two agents on the team. As the opponents pursue the player the top and bottom opponents exchange their aiming information (according to their policies). This causes each to aim above and below the player, respectively. By aiming high and low, respectively, they keep the player hemmed in as they close in. The team of opponents thus demonstrates a strategy by surrounding the player and blocking escape angles.

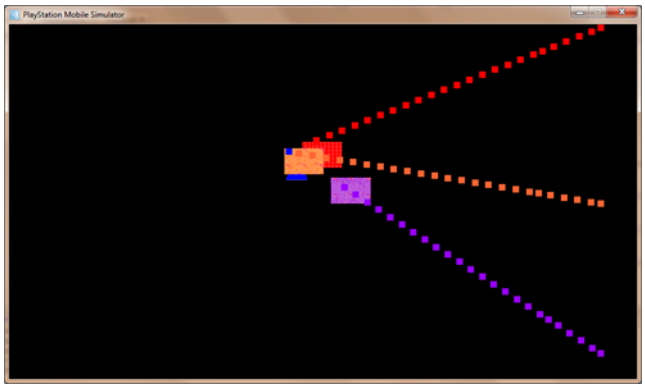


Figure 3.22: Simple Strategy Initial Steps

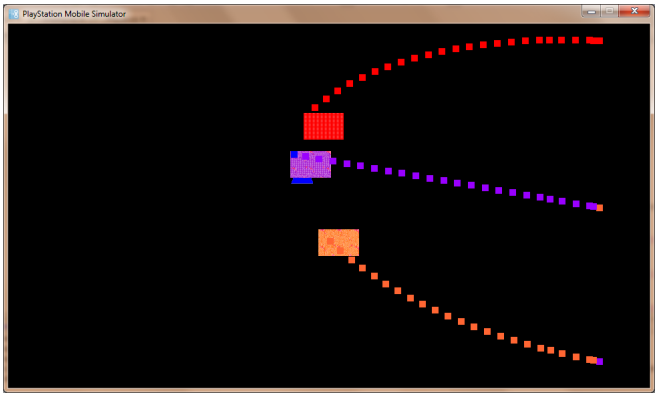


Figure 3.23: Intelligent Strategy Initial Steps

To calculate the proper pursuit lanes for the aim-and-miss policy the direct-pursuit angles were calculated first. This calculation was done using the Euclidean

distance as calculated via vector notation. The resultant vector is the direct-pursuit angle. This vector is then weighted against its opposite opponent's vector to derive a pursuit vector that takes the opponent just high or low of the player. To be specific, suppose the top opponent in the C-formation calculates its direct-pursuit angle to be $-30°$ relative bearing to the player. The bottom opponent then calculates its direct-pursuit angle to be $+40°$ relative bearing to the player. If these opponents turned to these headings they would be implementing the direct-pursuit algorithm. In the aim-and-miss algorithm each opponent adjusts its angle of pursuit by examining first its position in the formation, then its own pursuit angle, and finally its adjusted heading angle. In the example above the top opponent adjusts its heading by considering an equal-weighting formulation of its own direct-pursuit angle with the direct-pursuit angle for the bottom opponent. In this example the top opponent takes its own heading of $-30°$ and adds the heading of the bottom opponent ($40°$) to arrive at a heading of $10°$. This new heading angle insures that the top opponent will aim above the player and keep it from running into the player (and thus be easily avoided). The bottom opponent is also adjusting its heading to $-10°$ to aim below the player. These two will create the pincer effect to sandwich the player between the two outer opponents. As the top and bottom opponent approach from the edges they will stay outside the directly-approaching center opponent to surround and capture the player (Figure 3.24).

The tests were run by having individuals play the game with both policies, as dictated by the strategy. This user study was approved by an Institutional Review

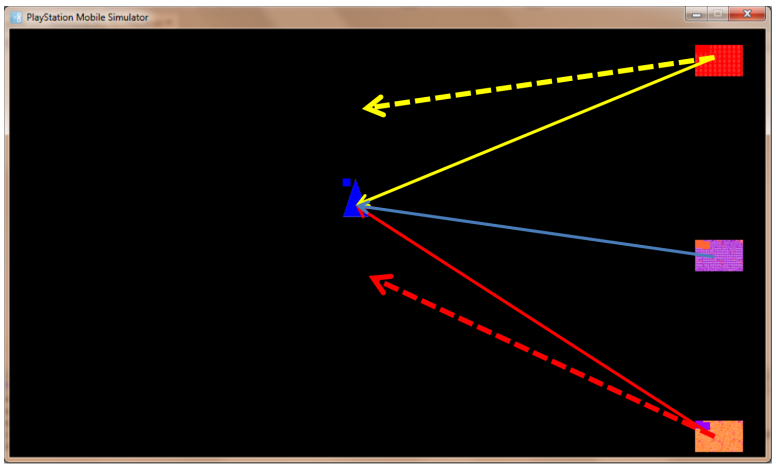


Figure 3.24: Pursuit Diagram of Aim-and-Miss Strategy

Board (IRB). They were given a short introduction to the game and oriented to the controls. After this, they played three games under each method. The opponent strategy they faced for the first three trials was determined randomly to remove experience bias. The participants controlled the player by moving with the directional arrows and worked to avoid the opposition. During each game there were traces placed on the screen to show the paths of the player and each of the opponents to show the pursuit paths. Each participant was then asked about their opinion of the two different policy schemas. The game was written for the PlayStation Vita and used the PlayStation Mobile (PSM) Development suite. The PSM uses C# programming.

3.9.3 Results

The study showed that the two strategies performed quite differently. The direct-pursuit strategy averages 21.34 seconds per trial (std dev of 9.52), with some games ending only after the participant grew tired of running around the screen. In these

latter instances it was clear they could have eluded the opponents as long as they wanted. The aim-and-miss strategy showed a clear advantage with an average of 5.63 seconds per trial (std dev of 1.68). The results of the various trials and their respective runtimes are found in Figure 3.27. While the runtimes may have varied, the difference did not the aim-and-miss strategy was the clear winner. The reason for the performance difference is quite clear as well. By looking at the initial pursuit paths of both schemas (Figures 3.22, 3.23) the difference is clear. The direct-pursuit strategy moves straight towards the player and the three opponents cluster together. Because their policy is the same they then perform as essentially one opponent. This quickly evens the odds for the player as it is easier to avoid one opponent than it is to avoid three. Figure 3.25 shows the pursuit paths after a long game play with this simpler strategy. The aim-and-miss strategy initial pursuit angles show that the top and bottom opponents approach the player from their respective sides and only curve towards the player as they get close. This shows the claw formation behavior quite clearly and exemplifies why this more strategic multi-agent approach performs better. Figure 3.26 shows this clearly as the game ends quickly because the player is trapped by the C-formation (the result of the team following the aim-and-miss strategy which implements the claw behavior by assigning policies to the team members) and cannot escape.

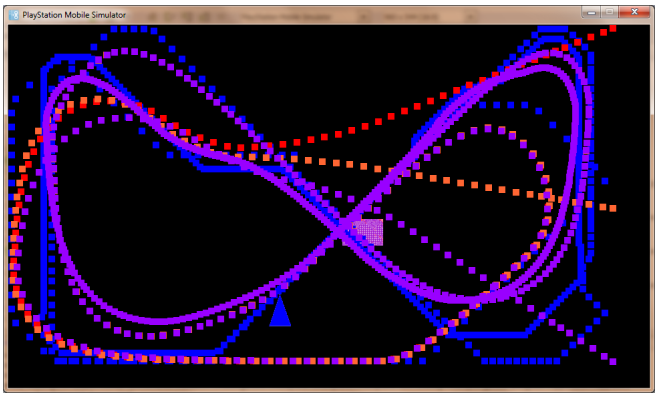


Figure 3.25: Full Run Tracking with Simple Strategy

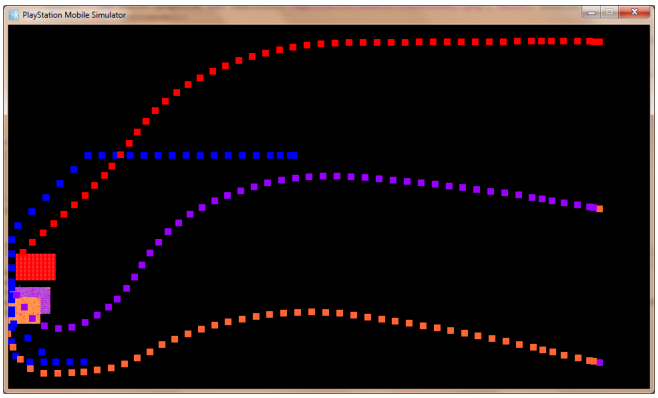


Figure 3.26: Full Run Tracking with Aim-and-Miss Strategy

3.9.4 Conclusions

We set out to show that one strategy, where each agent is assigned the direct-pursuit policy, cannot outperform another strategy, where the central agent is assigned the direct-pursuit policy and the other two agents are assigned the aim-and-miss policy. The first strategy shows the flaw of having multiple agents (uncoordinated activity) versus having a multi-agent system (coordinated activity) like that of the second strategy. The experiments showed clearly that the same simple game was considerably more difficult for the participants when the AI used a strategy that had them work

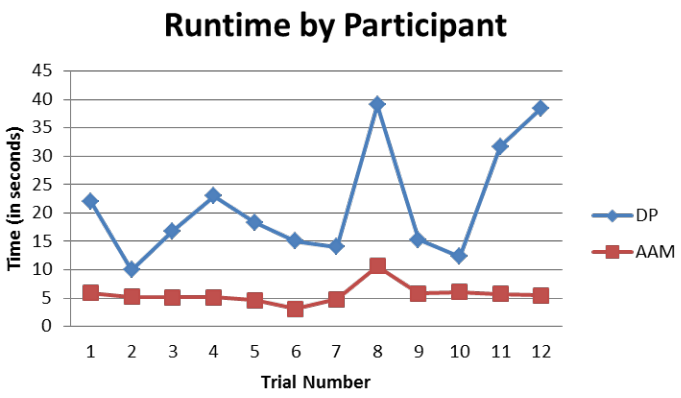


Figure 3.27: Runtime for AI Techniques

together to trap them. Listening to the participants talk about their experiences was quite enlightening:

> "The first game was easy, but after several rounds it became clear that I could trap the opponents by letting them almost catch me, then evade until I got tired. The second game, where they were working together to outsmart me [sic], was much more difficult. It felt like I was being stalked, like they were hunting me. I never even came close to getting away." *Participant 4*

The data was also convincing it makes a significant difference to have the opposing agents coordinate their efforts and maintain formations (Figure 3.27). This claw technique was confirmed in the traces and the gameplay it traps the player in a corner quickly and covers their exits. The trials confirmed the hypothesis (that agents working together strategically will outperform agents that are not). It was our goal to develop a simple test of these strategies and our game shows the promise realized.

In the future we wish to build on this research by examining additional formations to see their effect on the pursuit game. Additionally, we would like to examine other formulations of the weighted algorithms that perform the aim-and-miss scheme (the variables that control the angles, the amount of deflection, etc.). This game could also be modified to include obstacles, larger spaces, or 3D. There is more to be realized from the promise that this experiment has already shown and we want to build on this work to do so.

This experiment was designed to prove the importance of multi-agent strategy (like aim-and-miss) as compared to multiple-agent strategy (like direct pursuit). While the conclusion is not surprising, it is critical to the research presented herein that multiple agents can work together sharing a strategy. This cooperation is an essential element in implementing the team dynamics required in the larger experiments that follow in the next chapters. These experiments are designed to show multiple multi-agent teams sharing one environment. Each team will be its own multi-agent system. Each team will then work using its own strategy while comparing its strategy to those of the other teams. As mentioned in the initial discussion, the goal is to show that these teams improve their performance when they observe each other's strategies and either ignore them, adopt them, or adapt them for their own use. This experiment was an important milestone in the viability of such a system, and will be critical to the larger experiment's success.

Chapter 4

SiMAMT Framework

4.1 Overview

SiMAMT creates a realistic and complex environment in which the agents and teams of agents will act. The SiMAMT Framework is comprised of five distinct phases of processing, as shown in Figure 4.1. The first phase is the initialization phase called Strategic Modeling. SiMAMT is contingent on the ability to model strategy (i.e., to formulate complex systems of behavior into cohesive models). Once the models are in place the process commences with the Strategy Simulation module. As a note, the models are uncoupled from the framework, meaning that the models can change to suit the environment or to simulate different types of interactions, and the framework remains unchanged. The simulation module takes the Strategic Modeling data and sets up and initializes the simulation environment. Once the simulation is set up, it is put into motion, and it then produces data that is fed into the Evaluation Engine (EE) and the Strategy Inference Engine (SIE) for processing. The EE is measuring the intra-strategy performance (i.e., the performance of the various policies that make up the strategy) to make recommendations on policy changes

that need to be made. Similarly, the SIE is measuring inter-strategy performance (i.e., the performance of the current strategy vs. the perceived performance of other possible strategies and other team's performance). Once this data is consolidated and processed it is analyzed and evaluated. This evaluation is then forwarded to the Intelligent Strategy Selection Engine (ISSE) where a final decision is made as to the current strategy that should be in place given the evaluation. The cycle then repeats as the simulation continues until termination. This framework provides high-fidelity modeling of real-world interactions at each hierarchical level according to individual policies, behaviors, and group strategies.

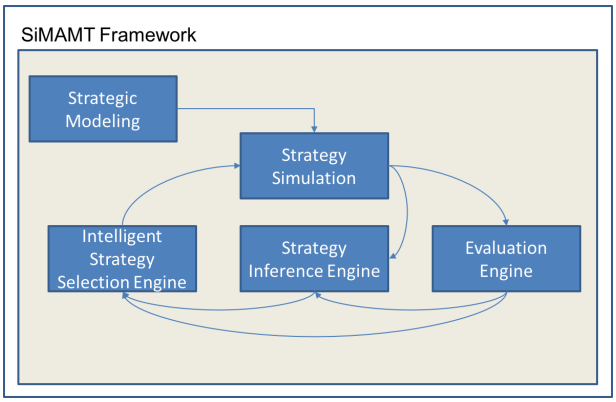


Figure 4.1: SiMAMT Framework

4.2 Strategic Modeling

SiMAMT utilizes a model-based approach to agent management, but the actual models are flexible. Previous iterations have used finite state machines and search tree implementations. When the strategies, and their associated policies, can be derived (or provided) probabilistically then they can be represented as either Finite-State

Automatons (FSAs) or Probabilistic Graphical Models (PGMs). These models can then be expressed in two ways: first, as a diverse sets of graphs for each such policy where the relevant walks in the graph represent strategic action chains (representing policies); second, as multiple isomorphic graphs where the weighting of the edges encodes the decision process. This means that multiple agents interacting within the same environment can use strategies (with their related set of policies) to execute their actions and, thus, act intelligently. In this scenario, then, it is possible to reverse engineer this strategic interaction based on observations of the actions taken by a particular agent (this is the work of the SIE). By comparing the observed actions with the probable actions of each policy, a belief network (BN) can be formed that leads the particular agent to predict the policy of another agent within the system. Combining the agent's observations of policies inferred from the observation of other agent's actions, the team leader can then infer the most likely strategy in play by the other teams in the scenario. In a system with increasing complexity, where calculating multiple factors may be time-prohibitive, the ability to match these candidate graphs (e.g., FSAs) with the currently forming belief network image (another graph) in soft real-time, or interactive time, can be challenging. As can be seen in Chapter 5, an approximate solution is available and can perform this matching in interactive time.

This work utilizes a FSA that allows for a similar action set for each agent but with customizable factoring (probabilistic progression through the model), shown in Figure 4.2 (this is the agent model). The agent has distinct phases that are competing for attention each turn. They can be thought of as Markov random fields with multiple

variables. This is reflective of the mindset of many real-world agents. The model starts in a move state and then is directed via the probabilistic pathing towards each of the possible states (e.g., an offensive, a defensive, an observation, or a movement phase). The weighting (factoring) of each agent action is probabilistically determined by the individual agent's policy. This policy is given to them from the strategy the team is following. Thus the agent considers their action with their own probability (based on their player type) which reflects their own 'personality'. This probability is then modified by the policy according to the overall policy goals. Finally, the team strategy weighs in on the probability. This gives the effect of individualized performance with overall short-term goals and team performance with match-wide long-term goals. *This is critical to the real-world performance of the system: it must emphasize individualized activities but constrain them (or at least influence them) by the team's overall goals (as realized in the team strategy).*

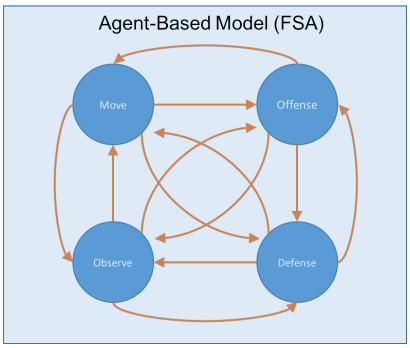


Figure 4.2: Agent Model as Finite State Automaton (FSA)

In particular, the FSA will model the basic behaviors and their probabilistic pathing. In the FSA shown in Figure 4.2, the agent starts in any of the states. By default, this is usually the move state. They will move through each of the

states in any order (though the model could be modified or customized to have the agent process each phase differently). In each phase, they have a probability of taking the designated action (i.e., invoking the underlying engine for each state of the agent model). This may mean choosing an offensive action (handled by the Targeting Matrix Model (TM)), choosing a defensive action (handled by the Threat Analysis Model (Threat)), making observations (handled by the SCAN Diagram Model (SCAN)), or choosing to move (handled by their Behavior Vector Model (BV)). The TM model scans through the 'lanes' visible from the agent's location to see if there are any opposing agents in view. If so, they are targeted and selected to be fired upon. The Threat model scans the same 'lanes' to see if there are any incoming threats (either from the agent 'taking fire' or from opposing agents approaching the agent's location). If there are threats, take cover. If not, choose the optimal posture (e.g., left, right, etc.) for the situation (e.g., if there are no targets on one side, switch to the other). The SCAN model searches the visible area for the agent to find any relevant observations (other agent locations, agent transitions, flag locations, teammate locations, etc.) and report them back to the FSA coordinating the agent behavior. They have a certainty of observation (meaning that there is not a probability on scanning, only on whether or not they will see something), both active (noting other agents in view) and passive (noticing zones from which they are receiving fire). These observations of the other agents positions, and most notably their transitions from position to position, are the key elements of the strategic inference during the simulation. Finally, the BV model is this particular agent's

subgraph of the Total Movement Dependency Diagram (the model of all possible movement for any agent in the simulation space, discussed in detail in Chapter 5).This model governs the agent's movement within their assigned area of the simulation space. Each of these models can be considered sub-models of the agent's FSA.

The basic agent FSA can then be expanded to show the sub-models (also FSAs) as shown in Figure **??**. These various models and sub-models are segmented for clarification in this figure. Each of these sub-models are also FSAs, though their specific models are not shown here. Moving up a level in the hierarchy, the agent models are assigned (as behaviors) to the agent by their policy. The expanded policy model, shown in Figure **??**, shows that the policy selects the behavior that is to be assigned to the agent from the set of available behaviors. This behavior is then assigned (i.e., implemented) in the agent. Finally, any evaluation data is sent back to the selection engine (this may include performance data on how this policy is working so far). The evaluation mechanism for the policy (i.e., it evaluates the quality and performance of the behavior) is the Value Function. This function may take on many forms, but it is a methodology of evaluating the efficacy of the behavior. This policy model can be compressed into a single element before going up to the next level of the hierarchy, as shown in Figure **??**. The policy is assigned to the agent by the strategy. The strategy model is shown in Figure **??**. As with the policy model, the strategy model first selects the given policy from the subset of policies available to the strategy. It then assigns that policy to the particular agent during the assignment phase. This phase loads the policy into the policy model for that agent. The Evaluation Engine

evaluates the performance of the policy and reports those findings to the strategy, and through the strategy to the Intelligent Strategy Selection Engine. This model can also be compressed, as in Figure **??**, so that the full effect of the strategy can be seen. In Figure **??** it can be seen that the strategy is assigning a policy to each agent on the team (as a reminder, the strategy can be viewed as a team policy). The strategy is assigned by the intelligence. The intelligence model is shown in Figure **??**. Like the other layers, it starts with a selection phase that chooses the strategy from the set of strategies available to assign to each team. The assignment phase performs this operation and puts the strategy into effect for the team. The evaluation is performed by the Intelligent Strategy Selection Engine (ISSE). The ISSE monitors the performance of the strategy as well as determining the most likely strategy that other teams are following. This evaluation is forwarded to the intelligence in case a new selection is needed. As before, this intelligence model can be compressed as in Figure **??** so that a larger view of the intelligence model can be viewed. This final view can be seen in Figure **??**. This figure shows a sample model with three teams. In this example, these teams would be cooperating since they share a central intelligence model. If there were another alliance (i.e., another set of teams) then they would have their own intelligence model. This final figure gives some perspective for the entire model used in the simulation.

This hierarchical arrangement of models allows for the processing of information at each level of the performance model rather than having to utilize one monolithic policy for the entire structure.

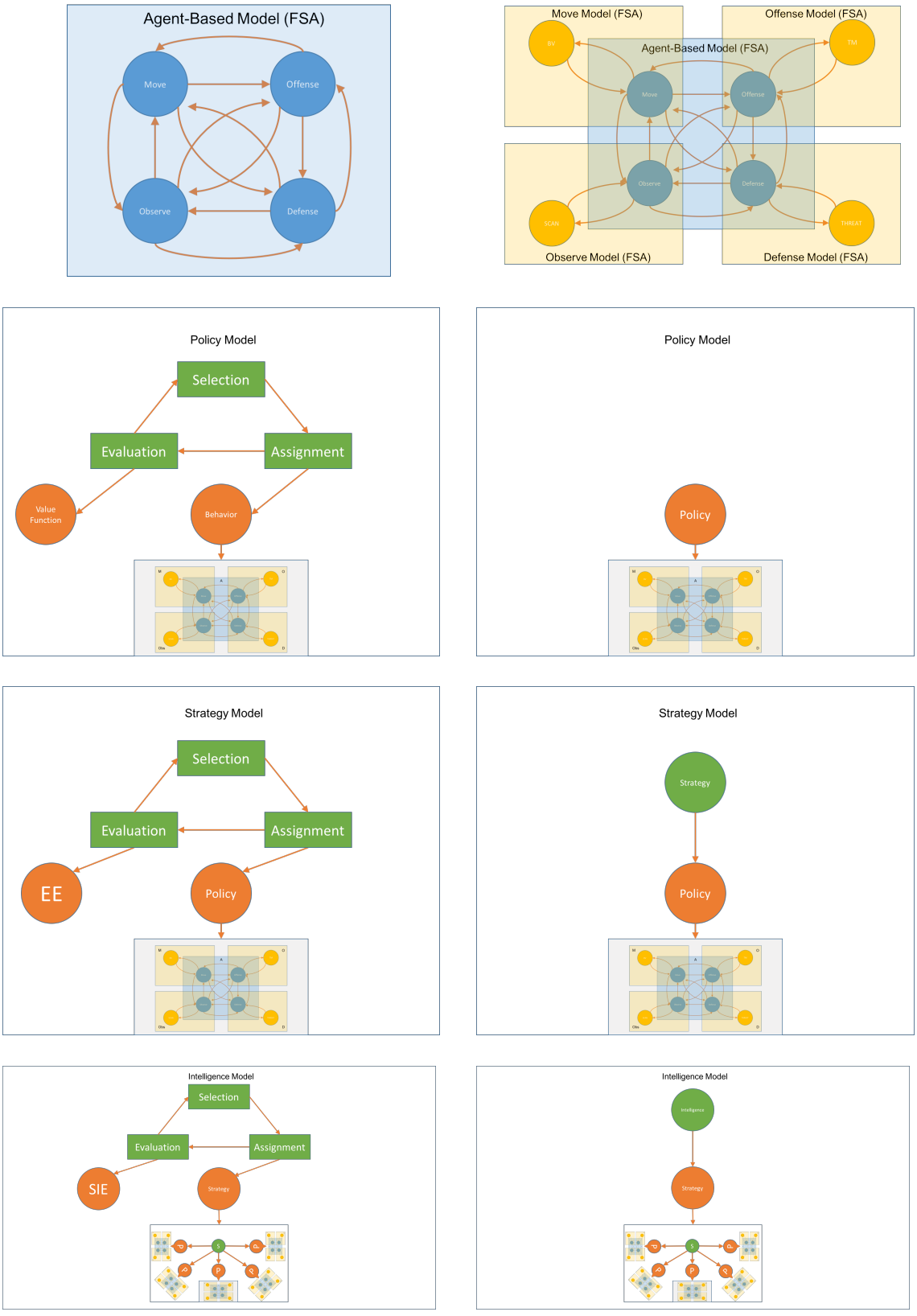


Figure 4.3: SiMAMT Framework Model

As previously mentioned, the models can be customized based on the scenario being simulated, but these figures show how the hierarchical modeling works within the SiMAMT framework. In Chapter 5 these models will used for strategy inference. In strategy inference each model forms a graph structure and the algorithm matches partial observations with known models to form a belief network about which of these various models the agents believe they are seeing. Building these graphs is demonstrated within that chapter, but it is based on the models presented here.

4.3 Strategy Simulation

Creating simulations for multi-agent multi-team interactions is a daunting task. It is non-trivial to compose a situation where each individual agent maintains their own 'personality' while still following the assigned policy dictated by a team's central command. Further, the complexity is inflated by ensuring that each of these agent policies is coordinated into a cohesive team strategy. Finally, peaking the complexity, is evaluating the performance of the team's strategy against other teams' strategies in interactive time. This is the work of this research, proposing SiMAMT, the simulation space for multi-agent multi-team engagements, and testing it. We will first cover the system and how well it models the virtual environment for strategic interaction. Second, we will deliver results from a practical test of strategy inference within such an environment using the SIE (Strategy Inference Engine).

In multi-agent multi-team scenarios, teams of agents are often trying to use the observed actions of the other team's agents to predict the behavior of the other

team. Strategy inference in such environments assumes that there is some underlying strategy that these teams are following, that such strategies can be inferred through observed actions, and that they can counter such strategies in interactive time by selecting a superior strategy. These strategies may be modeled using various graph structures such as trees, finite state machines (FSMs), or probabilistic graphical models (PGMs). To best determine which strategy a team is following it is necessary to match prospective models representing observed behaviors or policies with known models (each of which represents previously learned strategies and their related policies). Matching these models is difficult when their progression is probabilistic, the models can be homeomorphic (one is a subgraph of the other) or isomorphic (the bijection is true (i.e., homeomorphic in both directions)), and there are a variety of factors weighting each branch of the tree (Vazirani, 1989). Such intensive matching is taxing even on HPC systems, especially when interactive time decision making is key to the scenario. Further, the complexity of the multiple teams, each of which is running their own strategy, and multiple players, each of which is following a distinct policy customized for them by their strategy, make interactive time computation challenging. The results will show that the environment, which shall be of a satisfactorily complex nature, is successfully modeled as a multi-agent system (i.e., each agent following their own policy, each of these policies coordinated by a strategy at the team level) with multiple teams; further, the environment is able to be analyzed by the SIE, and that experiments verify both the execution of

strategies and their recognition (inference). Finally, the experiments will show that this inference leads to interactive time decision making within the match.

It will be shown that strategies offer significant performance enhancement to artificially intelligent agents, that strategies can be recognized in interactive time when complexity is limited, and that AI's utilizing strategy inference will outperform their originally superior opponents. In contrast, classical machine learning requires repetitive trials and numerous iterations to begin to form a hypothesis as to the intended actions of a given agent. There are numerous methodologies employed in an attempt to reduce the number of examples needed to form a meaningful hypothesis. The challenge arises from the difficulty created by the diversity of possible scenarios in which the machine learning algorithm is placed. Given enough time and stability a machine learning algorithm can learn reasonably well in a fixed environment, but this does not replicate the real world very accurately. As a result, strategies offer an opportunity to encapsulate much of this policy-space in a compact representation. They have to be learned as well, but the AI is learning smaller, individualized models rather than one large, monolithic policy. These models are also transmutable to another instance of a similar problem. Additionally, they can be pre-built and then modified to suit the exact situation. If these strategies are represented as graphs then they can be classified, categorized, identified, and matched. In particular, they can be represented as a variety of highly expressive graphs such as PGMs, FSMs with probabilistic progression, or other graph structures composed of complex elements. The SIE uses FSAs to build a belief network of candidate strategies from which it

selects the most likely strategy an opposing agent is using. Once created, this research scales this to work at even larger scales and with higher complexity.

Through the Strategy Simulation module, SiMAMT is coordinating the individual agents, each of which has their own unique features (e.g., speed, armament, etc.). It then groups these agents into any number of teams, as requested by the simulation configuration. These teams, then, have a leader (e.g., coach, captain, etc.). This leader first selects which agents on its roster to make active, then assigns a role and a policy to each agent according to the strategy for the team. This means that the list of roles (i.e., types of behaviors in the simulation) is distinct from the agents. Since the team leader is assigning roles to the individual agents on their team then matching the right agent to the right role is imperative for success. This may be a good matching or a poor matching (this variability allows for unique combinations of teams, roles, and agents, and thus adds 'role assignment' to the list of elements the simulation is modeling). Further, the individual agent's policy contains many variables about how the agent will act in the various game phases. This accomplishes the goal of maintaining individuality while imprinting each agent with their portion of the team plan. This could lead to individual agents breaking away from their orders and rebelling (where the agent's personal desires overwhelm their policy-assigned behavior), or it could lead to an agent's compliance (where the policy settings take precedence). This is an example of the many factors available for each agent, and reflective of the expressivity of the simulation (that claims to model complex strategic interaction between team members and between other teams). Additionally, the team

leader has more policies available than players, so they can reassign, switch, or drop policies during the match. The leader is also responsible for evaluating the overall performance of the team and, ultimately, deciding if the team should switch to another strategy. This process is addressed in the SIE portion of the research.

The Strategy Simulation follows the model loaded for execution. This model is hierarchical in that it incorporates a model built of models. Each layer of the hierarchy corresponds to a layer of the execution model in the simulation - strategy models, policy models, etc. For example, consider Figure 4.4 that shows an expansion of the Strategy Simulation with a sample model. This model shows the procedure for simulating the action of a single agent in a particular scenario. The agent is initialized by assigning all of their characteristics to an agent created inside the simulation (this agent will have distinctive qualities from other similarly created agents because of the variety of the characteristics and control variables even though the model may be the same). Once initialized, the simulation starts the agent control loop. Depending on the desired functionality, these various steps, shown here in a cycle, can be done serially or in parallel. The robustness of the simulation is that it follows the models, so it can work either way. Regardless of the mode of execution, the various steps are examined as indicated by the model. In this example, the agent makes a determination about whether or not it should move, take an offensive action, make observations, and take any defensive actions. There are a variety of factors that go into each of these decisions, but the overall view is that the strategy has initiatives, realized through the policies' behaviors, that direct the individual agent. The agent, however,

has innate characteristics that determine how likely it is to obey the orders directly without question or to act as an autonomous agent. The simulation progresses each agent through this cycle and relays commands to the agents and receives observations from the agents to pass along to the SIE.

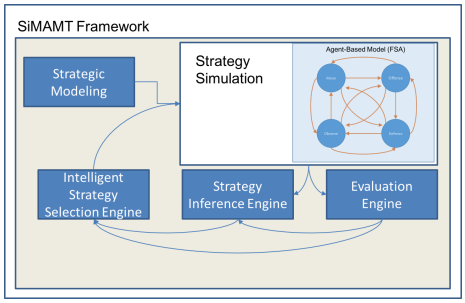


Figure 4.4: SiMAMT Framework: Strategy Simulation

Additionally, SiMAMT can also enforce additional constraints in the simulation, such as not being able to shoot or observe when under cover. It can set maximums or minimums, control overall conditions (e.g., weather, duration, etc.), and observe the overall performance of each team, each agent, and the simulation itself. The simulation engine can handle any number of players, active players, field positions, policies, and strategies as long as the data files provided detail each of them.

The simulation environment is constructed from data files. The files contain all of the configurable data that describes each aspect of the simulation in detail. The simulation engine is designed so that the inner workings of the engine are held within the system specifically so that it can be easily adapted to new simulations. As illustrated previously in the model-based diagram for SiMAMT, Figure 4.1, the data files that configure the simulation are held in the Strategic Modeling portion of the

framework. This section is separate from the execution-cycle elements of SiMAMT because it is the portion of the framework where the users can configure the models, parameters, and any other necessary configuration elements of the simulation. With the files configured and the scenario confirmed in the models, the framework then proceeds to the execution-cycle. Again, this portion of the framework utilizes those models that were input, so the inner workings of the framework do not have to be altered for simulation execution.

Once all of the data is loaded that configures the framework, the agents are then recruited onto teams. Once fully configured, the simulation assigns strategies to each team. These strategies then match agents with agent types (e.g., commander, sniper, etc.) and assigns an initial policy to each. Each player's policy then assigns them a particular behavior, complete with their movement diagram. Each player is then moved to their initial state (the home position on their movement diagram) and then the simulation begins.

The simulation is a discrete-time simulation where each agent (and each element of the environment) can both execute and receive actions at each interval. This is an agent-based simulation (or, agent-model-based, to be more precise). Algorithm 1 is utilized in the execution of the simulation. As previously mentioned, the FSA for each agent makes decisions at each interval. Based on the FSA shown earlier (Figure 4.2), the various sub-models are executed in the order needed (some steps can be skipped, done more than once, etc., depending on the model selected or the strategy in force). Using this sample FSA, each action can be considered in turn. First, the agent

examines its behavior to determine the next **move**. This is done probabilistically based on the behavior (that lists each next move and its accompanying probability). Next, the probability of a move is based on the Movement feature of the strategy. If a move is issued, the agent proceeds to the next state at the speed dictated by the policy. It should be noted that the speed with which they move determines their vulnerability during the move and the ability to aim during the transition. This provides an added realism to the simulation that is sensible (e.g., if an agent sprints, they are less likely to be hit, but less likely to aim correctly). The next action for consideration is **defense**. The probability is based on the Aggression and Posture factors. Again, defense may choose cover, and under cover the agent is unlikely to be hit but is unable to fire or make active observations. The next probable action to consider is **offense**. This probability is based on the strategy factor of Aggression and Posture and metered by the agent type and policy. The next action to consider is **observe**. This involves the same basic flow as the firing phase, but they are only noting which agents are visible and noting any transitions. If they are being fired upon then these positions are inferred to contain other agents and are so noted (this uses an inverse kinematic to determine reverse trajectories based on those positions observable from this current position). This completes the action selection FSA for this agent and the simulation moves on to the next stage.

To better understand the strategy factors involves examining them in context. For example, in the 5-vs-5 speedball paintball scenario, the Aggression speaks to how the strategy might overwhelm the policy to force a player to move where they might

Algorithm 1 *Simulation Engine* for Strategic Multi-Agent Multi-Team Simulations

Input: $\mathcal{I}$, a set of intelligences s.t. $I \in \mathcal{I}$
Output: I', the next Intelligence for I
{Initialization}
Read Configuration Files
Load Models
for all teams $\tau_i \in T_m$ **do**
  σ_j = AssignStrategy(τ_i, Σ_n)
  for all agents $o_k \in \tau_i$ **do**
    π_l = AssignRole(o_k, σ_j)
    MoveAgent(o_k, $\pi_{l_{p_0}}$)
  end for
end for
{Simulation Loop}
$Continue = true$
while $Continue$ **do**
  for all Intelligence I_a where $I_a \in \mathcal{I}$, $a = 0$ to $|\mathcal{I}|$ **do**
    for all Team $\tau_i \in T_m, T_m \subseteq T$, $i = 0$ to $|T_m|$ **do**
      for all Agent $o_k \in \tau_i$, $k = 0$ to $|\tau_i|$ **do**
        for all Phase $phase_m$ of agent model, $m = 0$ to $|phase \in model|$ **do**
          $Observations_{\tau_i}$ += Execute($phase_m$, o_k, π_{o_k}, σ_{τ_i})
        end for
        π_{o_k} = EE(τ_i, π_{o_k}, $Observations_{\tau_i}$)
      end for
      σ_{τ_i} = ISSE(τ_i, σ_{τ_i}, $Observations_{\tau_i}$)
      if $ISSE_{state} \in F$ **then**
        $Continue = false$
      end if
    end for
  end for
  Report Simulation Data
end while

not have otherwise. It may also overwhelm their choice to take cover. Likewise, the Movement factor controls how likely they are to move and also in which direction. For example, the players may start to retreat (move backwards through their move list) as the number of active players remaining on their team diminishes. The Distribution often overrides move probabilities to move players closer to each other or farther apart depending on this setting. The Posture factor controls fire aggression, movement aggression, and the agent's likelihood to stand and fire vs. retreat. Finally, there is a Persistence factor that keeps the players on their current policy or frees them to move to another policy. These factors, as well as the subset of policies, differentiate the strategies one from another. It is important to note that the complexity of this environment is possible even with each agent in the simulation following the same model (though with different probabilities). If the models varied, the SIE would be even more effective because the engine would have more diversity in the observations provided for inference.

The simulation is comprised of a series of engines that drive the action of the simulation forward. As the simulation runs it is gathering data and passing the data on to both the Evaluation Engine (EE) and the Strategy Inference Engine (SIE). The Evaluation Engine (EE) analyzes the policies in place for the team based on their strategy and decides if any players need to be assigned new policies for better performance. The SIE gathers data on what is being observed by the agents in the field and decides on the most likely policies and strategies being followed by the other teams in the system. Both of these engines produce output, and this output

(i.e., their evaluation) is fed into the Intelligent Strategy Selection Engine (ISSE). The ISSE takes this evaluation data and analyzes it in comparison to the most likely strategies in play by other teams. It then decides if a change in strategy is warranted. The ISSE makes the final decisions on strategies in play, strategy comparisons, and strategy decisions. Each is detailed in their own section next.

4.4 Evaluation Engine

The determination of the efficacy of current policies within a strategy is done by the Evaluation Engine (EE). To determine how the team of agents perform, the EE has several factors. This leads to an example EE tuple (Equation 4.1) that produces the next policy selected for a particular agent, π'_{o_i}. This formulation shows the factors, similarly scaled from 0.0 to 1.0, and the list of policies the strategy has access to (a subset of all available policies). The EE is a tuple comprised of the various factors: σ, the strategy, $\mathcal{A}$, Aggression, $\mathcal{M}$, Movement, $\mathcal{D}$, Distribution, Φ, Posture, and Ψ, Persistence. Each of these elements is defined later in the discussion of the application of these models.

$$EE =< \sigma, \mathcal{A}, \mathcal{M}, \mathcal{D}, \Phi, \Psi, \Pi_s >, \Pi_s \in \Pi \tag{4.1}$$

To understand this next step in the SiMAMT framework we will review some of the essential procedural elements that connect the strategies that we are examining. The evaluation will use each of these pieces to build a larger picture of each of the

elements that are in play in the current simulation. It is critical to revisit these concepts now that we have covered more of the detail so that this connection is more clear.

In this hierarchical policy application, a strategy is the next hierarchical step up from the policy, just as the policy is the next step up from the behavior, and the behaviors are the next step up from the actions. Where a policy assigns a behavior to a given agent, the strategy assigns the best policy to each agent from the set of available policies within the strategy. The EE selects the best policy for each agent based on the performance of the team. As previously stated, inside of the EE there are a number of factors related to the current strategy: aggression (how risk tolerant the team is), movement (from retreat to neutral to attack), distribution (how spread out the team will be), posture (from defensive to balanced to offensive), and persistence (how long to stay with policies until a change is made). These factors affect every aspect of the simulation and each agent's actions. In addition to these factors, the strategy component of the EE holds assignments. Each assignment is a matching of a player from the roster with a player type and a policy (actually, a series of probable policies that they will follow throughout the match). This means that each team can select a strategy which in turn selects a subset of particular players from the roster and assigns them roles and policies probabilistically. This gives the team a cohesive set of long-term goals with which to accomplish its mission. Additionally, the performance of each player is measured along with the overall performance of the team to decide if a certain player should switch to a different policy. All of this

combined gives a strong measure of the flexibility, customization, and complexity of the performance of the team.

The EE evaluates each policy and creates a belief network of possible policies (those available within the strategy) in an effort to determine the best policy for a particular agent based on the comparisons with its optional policies. This is the data that is forwarded to the ISSE for further evaluation.

The matching of models is non-trivial and worthy of separate consideration (see Chapter 5). These models can be difficult to analyze and match (Kumar et al., 2011), (Vazirani, 1989), especially when considering that two graphs under inspection could be homeomorphs or isomorphs of each other. The chapter referenced proposes an approximation algorithm for improving the total execution time of graph matching by eliminating any clear non-matches through novel constraints. One small example would be to determine if the maximum degree vertex in one graph has a match in the corresponding graph (once the graphs are sorted by maximal degree vertices). This algorithm has been proven to quickly eliminate a large portion of non-matching graphs without having to analyze every vertex, edge, and neighbor-matching.

For this belief network we chose to use loopy belief propagation (Pearl, 1988). This formulates the increased likelihood of candidates based on a message-passing paradigm wherein each observation sends messages to each hypothesis until either a time limit is met or a criterion is satisfied. In Equation 4.2 we see the messages passed from the variables to the factor, or for our purposes, the observations passed to the policy candidates. Next, we aggregate these messages around each hypothesis

(candidate policy), as shown in Equation 4.3. This process is then repeated at the next hierarchical grouping (i.e., the policies are now the observations and the strategies are the candidates).

$$\forall x_v \in Dom(v), \mu_{v \to a}(x_v) = \prod_{a^* \in N(v) \setminus \{a\}} \mu_{a^* \to v}(x_v) \tag{4.2}$$

$$\forall x_v \in Dom(v), \mu_{a \to v}(x_v) = \sum_{x'_a : x'_v = x_v} f_a(x'_{v^*}) \prod_{v^* \in N(a) \setminus \{v\}} \mu_{v^* \to a}(x'_{v^*}) \tag{4.3}$$

This leads to the convergence of these beliefs, for the variables, Equation 4.4, and for the factors, Equation 4.5. This convergence gives us the result of the likelihood of candidate policies within the system, and, thus, the candidate strategies that are comprised of these most likely policies. Granted, this is a double probability, and, thus, a double inference, but it is the method of hierarchical aggregation that best infers the strategy in-force within this simulation framework.

$$p_{x_v}(x_v) \propto \prod_{a \in N(v)} \mu_{a \to v}(x_v) \tag{4.4}$$

$$p_{x_a}(x_a) \propto f_a(x_a) \prod_{v \in N(a)} \mu_{v \to a}(x_v) \tag{4.5}$$

4.5 Strategy Inference Engine

In multi-agent multi-team scenarios teams of agents are often trying to use the observed actions of the other team's agents to predict the behavior of the other team. Strategy inference in such environments assumes that there is some underlying strategy that these teams are following, that such strategies can be inferred through observed actions, and that they can counter such strategies in interactive time by selecting a superior strategy. These strategies may be modeled using various graph structures such as trees, FSMs, or PGMs. To best determine which strategy a team is following it is necessary to match prospective models representing observed behaviors or policies with known models (each of which represents previously learned strategies and their related policies). Matching these models is difficult when their progression is probabilistic, the models can be homeomorphic (one is a subgraph of the other) or isomorphic (the bijection is true (i.e., homeomorphic in both directions)), and there are a variety of factors weighting each branch of the tree (Vazirani, 1989). Such intensive matching is taxing even on HPC systems, especially when interactive time decision making is key to the scenario. Further, the complexity of the multiple teams, each of which is running their own strategy, and multiple players, each of which is following a distinct policy customized for them by their strategy, make interactive time computation challenging. The results will show that the environment, which shall be of a satisfactorily complex nature, is successfully modeled as a multi-agent system (i.e., each agent following their own policy, each of these policies coordinated

by a strategy at the team level) with multiple teams; further, the environment is able to be analyzed by the SIE and that experiments verify both the execution of strategies and their recognition (inference). The experiments will show that this inference leads to interactive time decision making within the match.

The SIE, shown expanded in Figure 4.5, forms a belief network of probable strategies being observed. These observations, gathered during the simulation, are those opposing agents seen either standing still, moving, or taking an action that draws attention. In the partially-observable environment of the simulation framework, these are often unconnected and difficult to place into discernible patterns. The SIE takes these agent observations and aggregates them into a team view of the opposing team's movements throughout the simulation space. This is what forms the basis of the graph matching described in detail in Chapter 5.

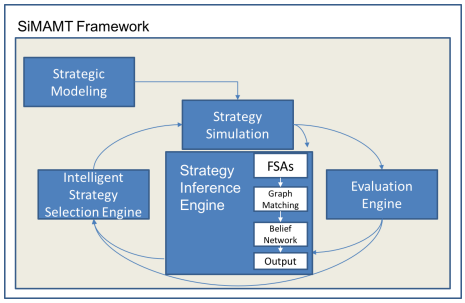


Figure 4.5: SiMAMT Framework: Strategy Inference Engine

4.6 Intelligent Strategy Selection Engine

The EE and the SIE take these aggregate belief networks and feed them forward into the ISSE. The ISSE then considers the evaluations of the performance of the current

strategy, the performance of the candidate strategies (most likely strategies in place based on the belief network), and the performance of optional strategies. Once this data is available, the inference engine recommends the most likely next strategy to implement based on these comparisons. If the current strategy is best, or within ϵ of the the best, then no change is made. If not, the best optional strategy is selected (either a better performing strategy from the evaluation or the best counter-strategy to the one suspected that the other team is following) and put in place. This means that the agents are all reassigned roles (assigned by the policies), behaviors (their movement through the field), and their decision making factors. The team instantly begins to realign itself with the new strategy and the process starts over again (though with more experience). The experiments show the results of the simulation (the veracity, complexity, and expressibility), the ability to implement and follow strategies (hierarchical policy networks), and the ability to infer opponent strategy and switch strategies in interactive time (by the work of ISSE).

As an important note, the recommendations from the EE are suggestions of needed policy changes, while the recommendations from the SIE are suggestions of needed strategy changes. However, both of these outcomes result in a shift of strategy. Recall that a strategy is a mapping of agents to policies, so shifting a policy for a given agent means performing a subtle strategy shift (to one for which that mapping is true). As a further explanation, consider that a strategy set is really a power set of all the variations of the policies available to it mapped to every possible agent within the team. This subtle shift, then, would be to one of these tightly coupled strategies

within this correlated strategy set. A change in strategy, in contrast, involves a full shift of strategy. When the SIE recommends a strategy change, the team now receives an entirely new strategy with a different subset of policies available to it, and, thus, a new power set of possible matchings of agents to policies. Utilizing these two methods, the ISSE can make subtle shifts of strategy by shifting to a related strategic application within the strategy set, or a much more aggressive shift of strategy by selecting a whole new strategy. In the former, there may only be one or two agents that shift to a new policy (because the strategy shift is within the power set, the other agents remain unchanged); in the latter case, every agent is assigned a new policy. This method of subtle or aggressive strategy change allows the ISSE to make fine-tuning adjustments to a team's performance or make a more radical change if something more drastic is needed. For example, if a team of five agents has four agents that are performing well but one that is underperforming, the EE would recommend a policy-shift for the single agent. The ISSE would, in that case, select another strategy from within the strategy set that has the same mappings for the four agents that are performing well but a new mapping to the recommended policy for the fifth agent. The result would be a subtle strategy shift to a strategy with this mapping where four agents remain unchanged and only the fifth agent is reassigned. In contrast, suppose that all five members are underperforming, and there are no variations of policies available within the current strategy that will perform well enough. In this case, the SIE would recommend a strategy change. The ISSE would then assign the recommended strategy to the team and all agent would receive the policies mapped

by that strategy. As previously indicated, the strategy sets have a default initial mapping that gets applied when first assigned.

The ISSE can be represented by another non-deterministic finite state transducer, defined generically as a tuple $(\Sigma, \Gamma, S, s_0, \delta, \omega, F)$. In this general case, Σ represents the input alphabet, Γ represents the output alphabet, S is the set of all possible states, s_0 is the initial state, δ is the transition function, ω is the output function, and F is the final set of states (accepting states). The currently utilized model, however, matches the one already offered in Chapter 3. The model from that chapter takes the generic formulation just mentioned and customizes it to the world of SiMAMT.

The data gathered is then utilized in the decision making process by the ISSE to evaluate the performance of the current strategy in relation to the performance of other strategies in play (those derived from the output of the earlier stages of the SIE as being most likely). This data is what enables the ISSE to make decisions. This much complexity makes any analysis equally complicated. As a result, the aforementioned approximation algorithms and model matching help reduce the overall compute time to bring this decision making ability to within interactive time constraints.

The hierarchical arrangement of the various elements used in this evaluation (the EE, the SIE, and the ISSE) provides flexibility and power. It is, of course, possible to accomplish this type of behavior analysis without implementing such a hierarchy, but the monolithic policies that resulted would be less flexible, less expressive, and significantly more difficult to learn (Matarić, 1997). We have not seen any such

algorithms working in interactive time within such complex environments (without supercomputing capability and reduced datasets). *This is an important claim for this research - strategic reasoning and inference offer increased flexibility, expressibility, and performance over monolithic policies.*

There are also multiple methods to vote for strategies, just as there were with the policies. The result is a belief network showing this comparison so that decisions can be made on the team's performance. The belief that a current behavior, policy, or strategy is in place or is being observed is directly correlated to the percentage match from the SIE and the EE, both of which pass their evaluations on to the ISSE (where these disparate elements are all combined to get a complete picture of the strategies in place). The ISSE then makes any and all policy, strategy, or overall variable changes that need to be made based on this aggregated data, evaluations, and recommendations. Once the policies and strategies have been evaluated the cycle continues back at the top with the simulation moving back into the forefront and starting the cycle over again.

4.7 Example Simulation 1

The SiMAMT framework collectively operates on the agents to move them through the simulation to produce the strategic interactions designed by the simulation engineers. An example of each of these elements working towards such a goal can be found in the Experiments section.

SiMAMT creates a realistic and complex environment in which the agent will act. In this example, we use the environment of 5-on-5 Speedball Paintball. This fast-paced version of paintball pits two teams against each other with a vast array of objects between them. All players must start in their home base and then progress from obstacle to obstacle while both avoiding enemy fire and attempting to eliminate the opponents. While there are flags for each team to capture and return to base, thus ending the match, the majority of the matches end via team elimination. There are several distinct phases of each round of the game but they can be reduced to offense, defense, and end game. Most of the teams will start out on offense and then shift to defense based on progress and eliminations. The decision to switch to end game is necessitated by the elimination of the majority of teammates. SiMAMT utilizes a model-based approach to agent management, but the actual model is flexible. Previous iterations have used finite state machines and search tree implementations. As mentioned in the introduction to this section, and repeated here, this work utilizes a FSA that allows for a similar action set for each agent but with customizable factoring (probabilistic progression through the model), shown in Figure 4.6. The agent has distinct phases that are competing for attention each turn. They can be thought of Markov random fields with multiple variables. This is reflective of the mindset of many real-world paintball players. The model starts in the move phase and then is pulled via the probabilistic pathing towards each of the possible states: a move, a cover, a fire, or an observation phase. The weighting (factoring) of each agent action is probabilistic on the individual agent as viewed through their

policy. This policy is given to them from the strategy the team is following. Thus the agent considers their action with their own probability (based on their player type) which reflects their own 'personality'. This probability is then modified by the policy according to the overall policy goals. Finally, the team strategy weighs in on the probability. This gives the effect of individualized performance with overall short-term goals and team performance with match-wide long-term goals. *This is critical to the real-world performance of the system: it must emphasize individualized activities but constrain them (or at least influence them) by the team's overall goals (as realized in the team strategy).*

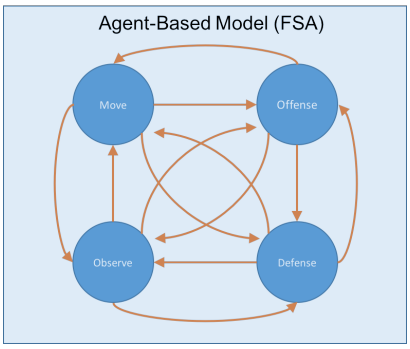


Figure 4.6: Agent-Based Model (FSA)

Again, so that this section can be read as a stand-alone example, the introductory material is repeated here. SiMAMT is coordinating the individual agents, each of which has their own unique features (e.g., speed, armament, etc.). It then groups these individuals into any number of teams, as requested by the simulation. These teams, then, have a leader (e.g., coach, captain, etc.). This leader first selects which players on its roster to make active, then assigns a role and a policy to each. This means that the list of roles (i.e., types of behaviors in the simulation) is distinct from

the agents, so this may be a good match or a poor match. Further, their policy contains many variables about how the agent will act in the various game phases. This accomplishes the goal of maintaining individuality while imprinting each agent with their portion of the team plan. This could lead to issues of rebellion, where the agent's personal desires overwhelm their policy, or compliance, where the policy settings take precedence. This is an example of the factors available for each agent, and reflective of the expressivity of the simulation. The team leader has more policies available than players, so they can reassign, switch, or drop policies during the match. The leader is also responsible for evaluating the overall performance of the team and, ultimately, deciding if the team should switch to another strategy. This process is addressed in the SIE portion of the research.

With respect to this implementation (this particular exemplar simulation), each team has their own side with a number of obstacles of varying size, protection, and position. While both sides are similar, they are mirrors of each other, so there is a right and left hand portion to each side of the course. Figure 4.7, an official field diagram from PSP Event held in Phoenix, AZ (Warpig.com, 2016), shows one half of the field with the positions labeled. The movement pathway of one policy is shown with solid lines, and the lanes visible from that final position are shown with dashed lines. This research implements the field in full detail as read from a file. This means that to play on another paintball course the coder need only create a new field file (the master file that is read during the setup of the simulation). The field is comprised of each of the object positions by noting where a player could stand in relation to

them (e.g., left, right, and cover each have their own distinct position number). In addition, it must be denoted which locations on the field are visible from a certain side of each obstacle. These are called lanes, and they represent the locations that a player can see during the observation phase as well as the players they can target during the firing phase. This flexible mapping feature, combined with a rich modeling environment, make the simulation robust and flexible.

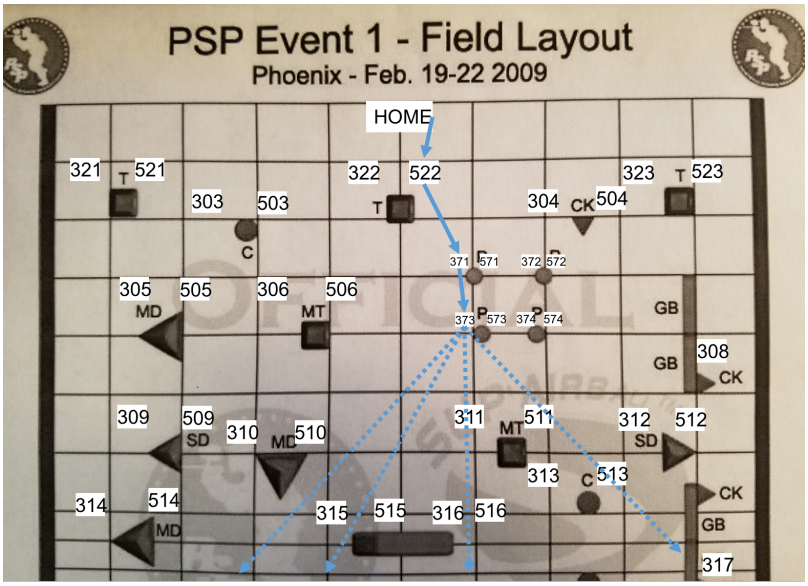


Figure 4.7: Official PSP Field

Additionally, SiMAMT can also enforce additional constraints, such as not being able to shoot or observe when under cover. It can set maximums or minimums, control overall conditions (e.g., weather, duration, etc.), and observe the overall performance of each team, each agent, and the simulation itself. Some additional settings in this particular simulation: no blind-firing (an agent must see another agent before it is allowed to fire), no stacking (no double-occupancy of any one side or a particular

obstacle), and match stats (e.g., the number of hits before elimination, size of each team, etc.). The simulation engine can handle any number of players, active players, field positions, policies, and strategies as long as the data files provided detail each of them.

4.8 Example Simulation 2

The second example will serve as a testbed for the various algorithms and implementations of the learning and recognition system. This game will take place in an arena, an area designated to bound the game into a limited domain. This arena, scaled for the agents, will be in a virtual box that is approximately 10 ft square. This arena will contain three goal areas, marked as 1, 2, and 3 (as shown in Figure 4.8). Within each goal area there will be two zones, designated by color and shape. The smaller red circle will be contained within a larger green circle. These two circles will represent the general target area (the green) and the specific target area (the red). These areas will factor into the scoring values for the game, with the specific target 'paying' double the reward of the general target area.

Any other areas, outside of these circles, will be considered a 'no-man's-land', and any players caught in this area will be penalized. At regular periods there will be a scoring signal that grabs an instantaneous image of the arena and generates a score based on the current location of the players. There will be three teams of agents (the 'players') introduced into the arena. They will have limited knowledge about the game other than the idea of the scoring zones and the regular (but unknown) period

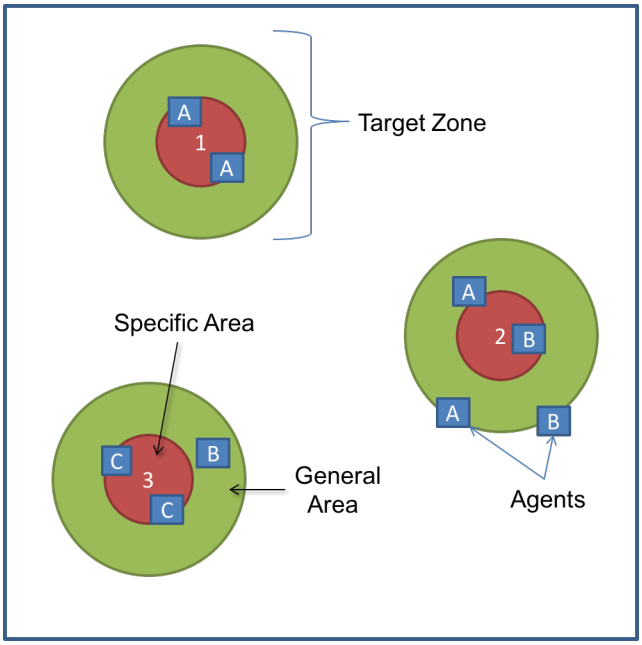


Figure 4.8: Game Arena Diagram

of the scoring. These teams will seek to distribute themselves around the arena to accomplish several goals: find the zone with best reward, get the most agents they can into the highest scoring area within that zone, distribute other agents to other scoring zones (since they cannot all fit into any one zone), and prevent others (where possible) from entering these same zones. The game will be scored progressively and the other teams will either know the score of the other teams (Scenario 1) or will have to infer the other team's score (Scenario 2). Initially these teams will be homogeneous, with each team having the same abilities and limitations. In further research the same game will be applied to teams comprised of different styles of agents with varying abilities and limitations, and then to hybridized teams where the team itself is comprised of different types of agents.

By construction, this game is a stochastic, repeatable, and partially-observable game. It is also allows for simultaneous actions. While the focus agent can observe their own belief network completely, they cannot view the belief of the other agents (Russell and Norvig, 2003). As such, game theory is essential for understanding these complex agent interactions.

This game is analogous to a free-market competitive environment where multiple companies compete, with varying resources, for a fixed pool of customers. There are various strategies that the teams can choose to implement, and these options may be restricted by the resources available to each team. For example, suppose Team A has better financial backing, and thus is able to spend more time on market analysis and research. They will then place their agents in the best regions based on this research. Another team, say Team B, may have more limited budgeting for these same areas of market analysis and research, but they could adopt a strategy of imitation (i.e., they place their agents in the same areas as their larger competitor, Team A). Team C, on the other hand, may have a strategy of avoidance (i.e., they seek to fill the gaps, placing their agents in uncovered regions) and leverage their smaller footprint and reduced expenses to reach into these smaller markets. When these three companies, each with their own strategies for expansion and profit, interact in a large scale the effects they have on each other may cause them to reconsider their strategy and either imitate another team's strategy or innovate a new strategy. There may be some benefit to hybridizing strategy, or stratifying strategies, to achieve more balanced results. These strategies will be discussed in more detail in the sections that

follow, but are introduced as an inducement to the larger impact of this game and how it can be applied to real-world scenarios.

A strategy in this game contains policies that follow the principles of the game. For example, Team A will have a strategy that has simple policies: target, transport, defend. Another team, Team B might have another strategy with different policies: distribute, transport, assess. From a general perspective the policies would be defined as follows:

- *Target* : Select the single optimal location from all available locations.
- *Transport* : Move to the desired location(s).
- *Defend* : Keep others away from this location.
- *Distribute* : Select multiple targets maximizing coverage.
- *Assess* : Compare current expected rewards.

This representation shows immediately the Case Base Reasoning (CBR) potential for this hierarchy. The ability to reason about the interaction of multiple teams and players at this level provides a 'coaches' ' view of the game. By way of example, CBR can be demonstrated by some logic like that found in Equation 4.6. This will greatly aid in the scaling of complexity with such diverse strategic interactions.

$$if\left(assess_i < assess_j\right), distribute_i, else\left(defend\right) \tag{4.6}$$

This will allow for a higher-view management of teams and agents within the game without the encumbrance of massive calculations. Before this type of representation is sensible, it is imperative to formalize some examples.

For example, the target policy, Equation 4.7, and the transport policy, Equations 4.8 and 4.9, can be thought of as selecting the location, v, with the highest expected reward, E from the list of locations, V and then calculating the lowest cost route to the desired location. To calculate the lowest cost route to v' the algorithm creates a subset of all paths P that contains all possible paths $p_{v'}$ that lead from the current location to the target location. Thus the cost function returns the traversal cost for each possible path in the subset of paths that lead to the target location, and the *argmin* delivers the lowest cost to reach v'. This gives a' as the action to take that leads to the next step along the lowest cost path.

$$\pi_{target} : v' = \underset{v}{\text{argmax}}(E(v)), \forall v \in V \tag{4.7}$$

$$\pi_{transport} : a' = moveTo(v') \tag{4.8}$$

$$moveTo(v') : \underset{p}{\text{argmin}}\, Cost(p, v'), \forall p \in p_{v'} \tag{4.9}$$

This policy, which is given rather than learned, directs the agent to the optimal location. This policy, if it were not provided, could also be learned through

experimentation by any of the Expectation Maximization (EM) methods. It is important to this research that it be clear that learning one of these policies, like the transport policy, will be easier and take less time than learning some all-encompassing policy that covered target, transport, and defend. Strategy is implemented to direct the agent and team actions by selecting the optimal policy to have in place.

While these policies and the overarching strategies will provide a simpler method for reasoning in single agent environments, the larger benefit comes in multi-agent settings where competition, cooperation, and team dynamics need to be considered. This research seeks to perform the experiments to confirm this hypothesis, define the finite state automatons for recognition and the belief network for inference, and finalize the opportunities afforded by higher-level case based reasoning.

Chapter 5

Graph-Matching Approximation Algorithm for the Strategy Inference Engine

5.1 Movement Dependency Diagrams

5.1.1 Introduction

Central to understanding how the framework can infer strategy is recognizing the key role of Movement Dependency Diagrams (MDD). These diagrams, overviewed earlier, show the possible movements for agents within the field. All movements within the field of the simulation can be viewed as walks along a graph, the Total MDD (this graph is a model of all possible moves for any agent within the simulated arena). MDDs are subgraphs of the Total MDD. As the graph is cyclical and bi-directional there are an infinite number of such walks, but the strategy is predicated on purposeful walks (moving towards a goal).

5.1.2 Motivation

SiMAMT provides a large-scale view of the arena of engagement, whether that is a theater-wide battlefield, a friendly game of soccer, or the complex world of

commodities trading. This view is hierarchical and is comprised of sub-views within each layer of the multi-tiered model. At each layer, the intentions of the individual elements at that particular layer need to be understood. Primarily, this is the purpose for creating strategy-based systems. The goal is to both simulate movement within each layer and understand such movement from other agents. To do so, Total MDDs are created. The motivation is to provide a structural representation of the interactions among agents being considered at any particular level of the hierarchy. The subgraph MDDs can be derived from observation via machine learning (watching agents within a particular field operate), a priori knowledge (domain knowledge presented as models), or some hybridization of the two approaches. These structures can then be utilized to encapsulate one type of agent behavior within a space, thus producing the graph that both describes and controls the agents movements within that space at some particular level of the hierarchy.

5.1.3 Approach

To create these structures the particular agent interactions must be studied. As mentioned, this may result in clear (or desired) models that agents can follow, or the observation may require advanced pattern recognition to discover the underlying models. Building these structures can be handled iteratively once a model is provided that governs movement within the layer. For example, given the model of a racetrack, the individual cars on the racetrack can be understood to move within its confines and in accordance with its directions. The general principle is to work to create

independent models at each striation to provide maximum flexibility to the overall model and to distribute the load of simulation and learning within these environments to each layer (rather than an unwieldy monolithic model that would struggle to learn in such diverse environments and have difficulty simulating behavior without super-computing capacity). *This is an important claim for this research - strategic reasoning and inference offer increased flexibility, expressibility, and performance over monolithic policies.*

The models utilized in SiMAMT have, as their atomic elements, states. These states, as previously defined in Chapter 3, are encapsulations of positions within the simulation arena. Each of these positions holds the actual location in the arena and a posture, or alignment, with respect to that location. These states represent a degree of coarseness or granularity depending on how they are defined. At some level they may only define large areas of movement, while at other levels they may represent very fine-grain movements. For example, consider a map that shows every airport within the United States. The great majority of these airports are very small and only allow for small, private planes to land there. If the goal was to create a Total MDD for air travel within the US then each of these myriad airports would be a state, and thus become a vertex on the Total MDD. Every airport connection would be labeled with an edge (i.e., a movement, defined as a change in state from one to another). This resultant Total MDD would be very fine in its resolution and thus very detailed. However, if the goal were to plan flights for large, commercial airlines then the resolution could be moved to a much more coarse representation. The resultant

Total MDD would be simplified. Here each vertex would be a state that represents a large commercial airport, and the edges would be those flights that connect these large airports. As an example of this type of granularity, Figure 5.1 shows the number of airports in a local region. When the scale is increased the coarseness increases, like in Figure 5.2 that shows regional airports. When the scale becomes national, like in Figure 5.3, the coarseness reaches it maximum.

There are two terms being represented here: first, the idea of granularity; second, the concept of connecting physical location within the simulation arena with vertices on the Total MDDs. Continuing this example, the entire listing of airports could be the Total MDD, and the subgraph of large commercial airports one of the MDDs. This fits well with the desired purpose of the SiMAMT framework. Here the small private airplanes can be modeled in their own subgraph of smaller airports, each bound to some finite range of distance of travel. The simulation could then easily move these airplanes from one airport to another according to their model while providing each plane with its own behavior. Further, many such airplanes could be given their own MDDs that reflect their area of travel and unique behaviors. While these many small airplanes are moving about on their MDDs, a larger MDD for commercial airplanes could be utilized by the simulation to move commercial traffic. Additional examples will be provided to show this concept of how Total MDDs are created and how MDDs can define a particular behavior within the larger Total MDD.

One further point of connection, based on the airport example mentioned, is that these MDDs represent particular behaviors, and are thus assignable to agent via

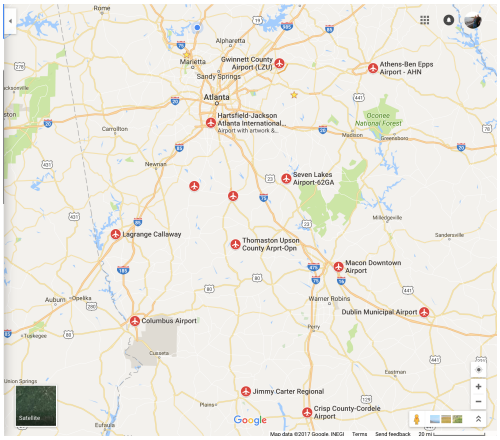


Figure 5.1: Airports in the US: Local Granularity

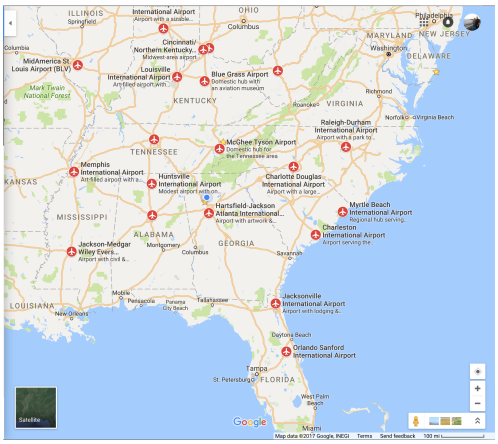


Figure 5.2: Airports in the US: Regional Granularity

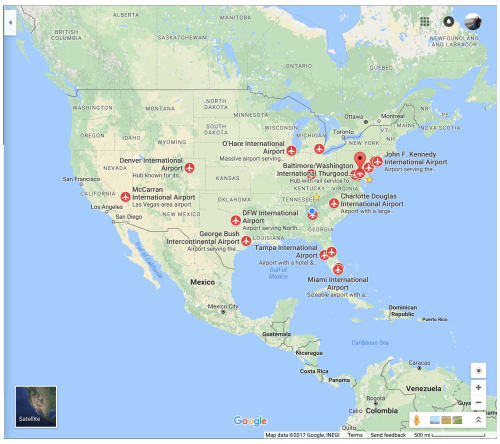


Figure 5.3: Airports in the US: National Granularity

policies. Figure 5.4 shows the Total Movement Dependency Diagram for the local region airports. This Total MDD could be broken down into MDDs, as shown in Figure 5.5 where each color represents a different MDD. In this figure each MDD could represent a single agent or a team of agents that cover a specific region and transit this MDD as part of their daily routine. Also, teams of airplanes (i.e., airlines) could utilize a strategy that assigns such policies to each plane within their organization. In this manner, each plane within the company would have its own MDD to control its movements within the simulation space because the airline strategy would assign a policy to each agent (i.e., airplane), and each policy would map an agent to a behavior, like that shown in Figure 5.5. This behavior becomes the 'roadmap' for movement within the simulation space. This is of two-fold consequence: first, multiple teams of agents can be modeled and controlled at varying levels of granularity; second, a cogent observer could, by monitoring the movements of another team of agents, infer the strategy that they are following because their pattern of behaviors would match known models. These two consequences comprise the bulk of this section of the research.

As another example, consider some sample diagrams from the sports world. Considering American Football, there are several instances of movement dependency diagrams for certain situations. Figure 5.6 shows an American football field with the ball positioned around mid-field. The area of interest centers around the ball, shown in Figure 5.7. The Total MDD would be every definable location within the field. While this could be infinite, it is reasonable to use Voronoi points to aggregate those

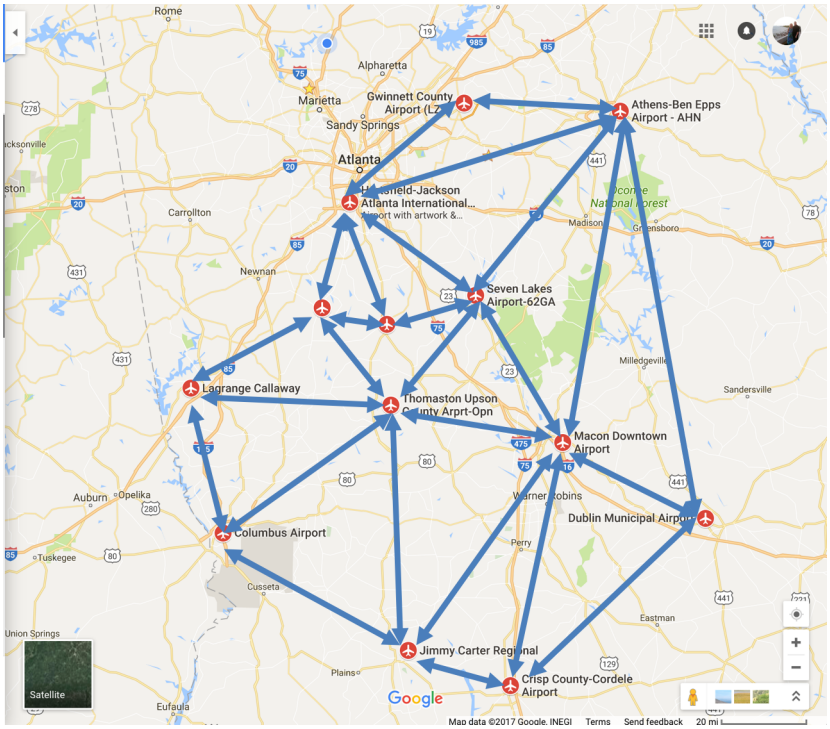


Figure 5.4: Total Movement Dependency Diagram for the Local Region

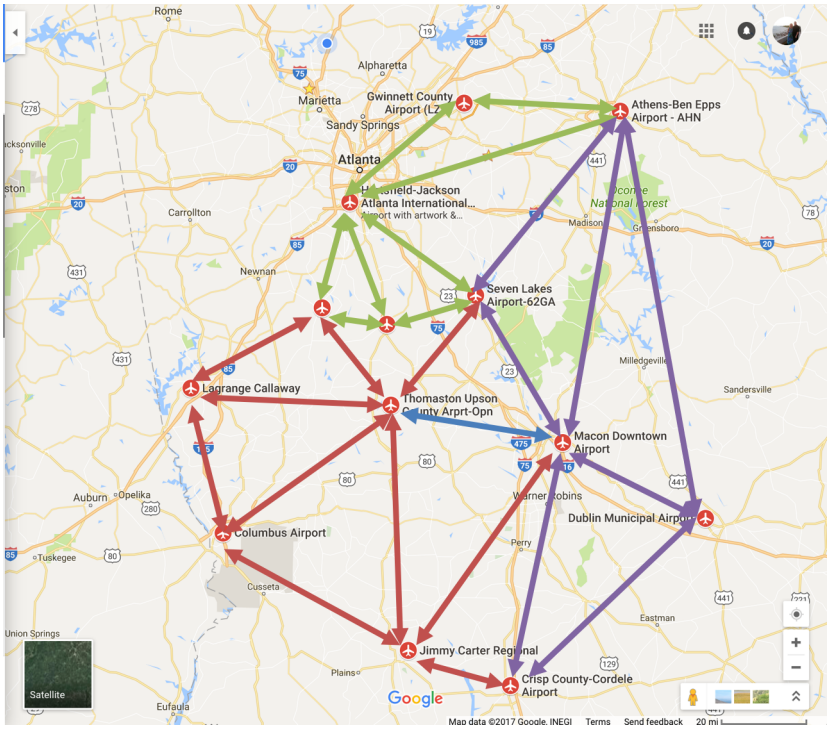


Figure 5.5: Movement Dependency Diagrams for the Local Region

points that lie within a certain range to a centroid for the region (Figure 5.8 shows the region of interest defined by Voronoi regions). On the football field or a soccer filed this may be a three-foot circle, or perhaps a five-foot circle. For football, we can use three-foot circles and spread them across the field at three levels from the center of every play (as shown in Figure 5.9). It should be noted that for these plays the orientation is to a central location, namely, the football, rather than the field itself. This results in a series of positions facing the ball on either side stretching from one side of the field to the other. There would be a second row of similar positions on each side, facing the center, behind that row. Finally, there would be a third row behind that one. Each of these positions would be vertices in the Total MDD, and there would be edges where there were moments from position to position (because these positions in the diagram are adjacent, the arrows between are negligible, but the Total MDD still uses these infinitesimal edges for movement). Using this Total MDD, a play could be configured that shows players in some of these positions (Figure 5.10). A strategy for the team would show each of these players with their own MDD (assigned to them as their behavior, provided by the policy), as seen in the examples that follow.

There are several real-world examples of movement diagrams provided. First, consider Figure 5.11 that shows the movement for each player for a particular zone blocking scenario. Second, Figure 5.12 shows an offensive play designed to allow the ball carrier to move through the defense with the aid of his blockers (each of which have their own assignment, as shown in the MDD. Third, there are two diagrams

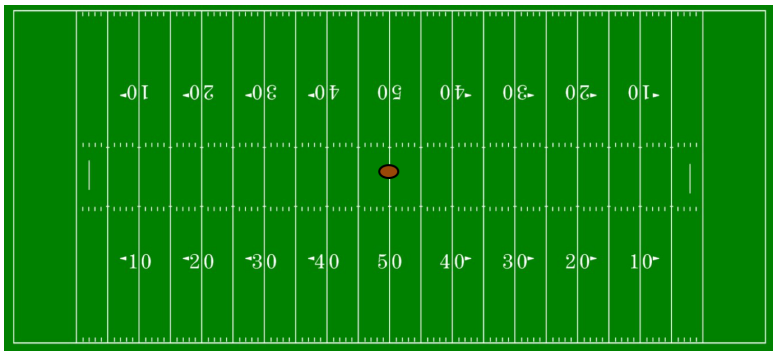


Figure 5.6: American Football Field

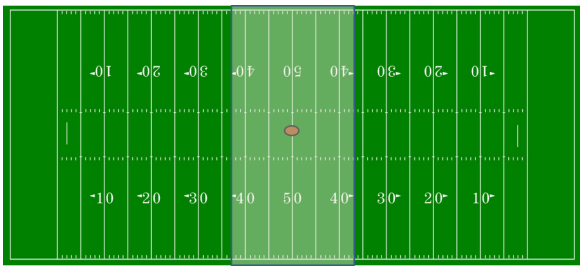


Figure 5.7: American Football Field: Area of Interest

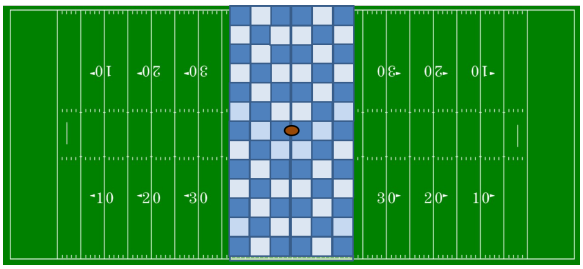


Figure 5.8: American Football Field: Voronoi Regions with Area of Interest

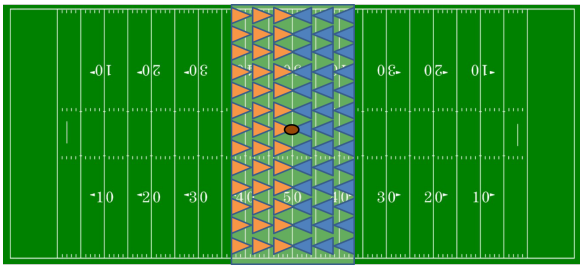


Figure 5.9: American Football Field: Positions Derived from Voronoi Regions

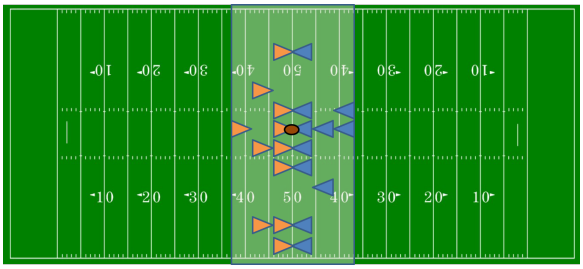


Figure 5.10: American Football Field: Sample Play showing Players in Derived Positions

that show passing routes, Figure 5.13 and 5.14, for the half-back and tight end, respectively. Each of these diagrams shows how a strategic plan may be put into motion by diagraming the actions of each cooperative agent on the team. Further, it shows the synergistic power of each agent on a team working together on a cohesive plan to accomplish what no single agent could have accomplished on their own.

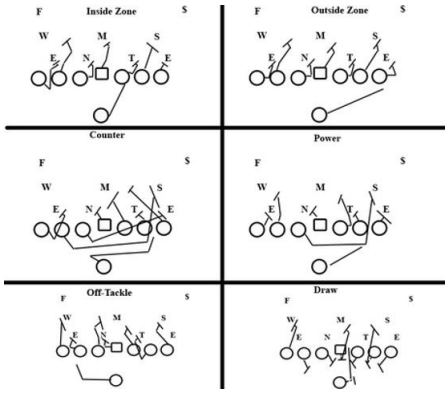


Figure 5.11: Football Play: Zone Blocking (BPAA, 2017)

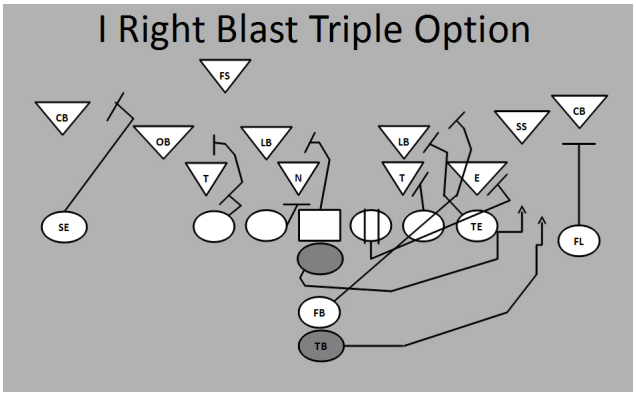


Figure 5.12: Football Play: Offensive (Tutorials, 2015)

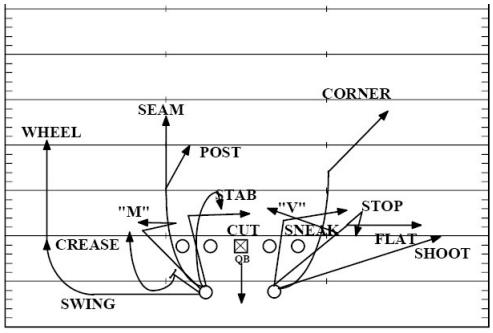


Figure 5.13: Football Play: Passing Routes for Half-back (Staff, 2015)

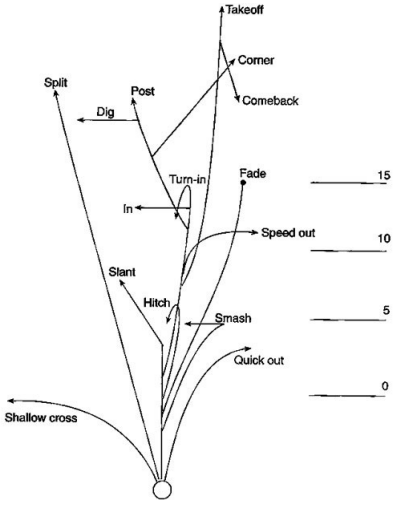


Figure 5.14: Football Play: Passing Routes for Tight End (Staff, 2015)

As an application illustration, there are two scenarios to examine. First, consider a soccer field, as shown in Figure 5.15. This field has a number of positions where an agent can be located, and the agent can move to any of these locations. Looking at the graph created by the exploration of this state space through every possible action results in Figure 5.16, where each vertex is the result of a movement and each edge the moment itself. Extracting this graph from the field diagram results in the state space for this instance, or the final Movement Dependency Diagram, as shown in Figure 5.17.

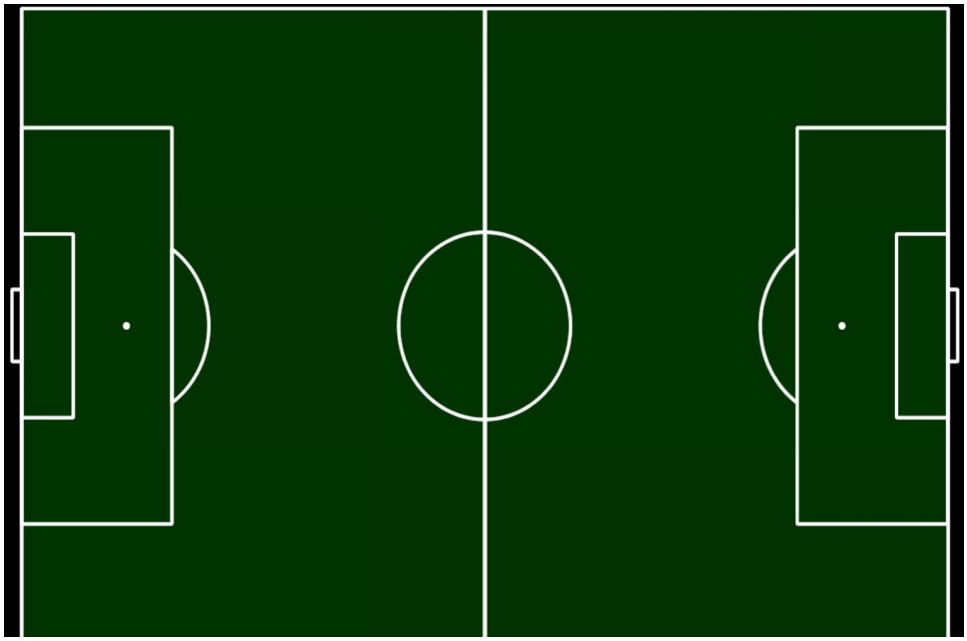


Figure 5.15: Soccer Field

Next, consider Professional Speedball Paintball. The field is shown in Figure 5.18a, with the movements mapped in Figure 5.18b, and the resultant Movement Dependency Diagram shown in Figure 5.18c. By examining the strategy, comprised of policies that map to behaviors, and behaviors that generate movements, it is possible

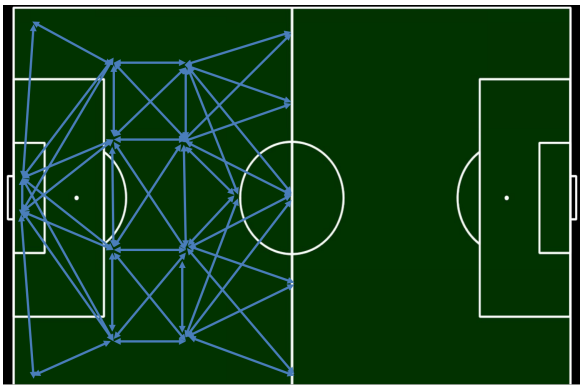


Figure 5.16: Movements for Soccer

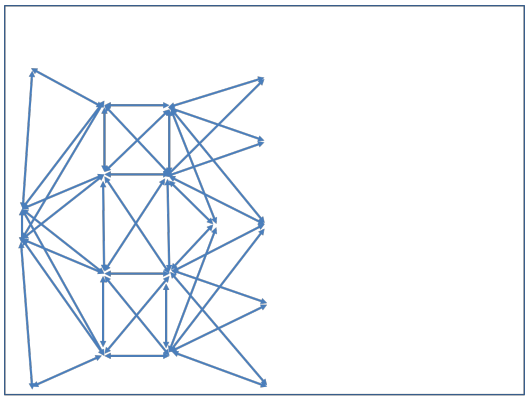


Figure 5.17: Movement Dependency Diagram: Soccer

to use the strategy to develop the Movement Dependency Diagram for each scenario. To do so, each policy available within the strategy is assigned to agents, then each agent receives a behavior from the policy, then the behavior iterates through every possible movement within the simulation. By tracing these movements, the entire diagram is created. By iterating through each possible strategy, via the ISSE, the entire Total Movement Dependency diagram can be created by tracing each of the Movement Dependency Diagrams based on the behaviors.

All movements are then viewed as walks along this graph, the Movement Dependency Diagram. As the graph is cyclical and bi-directional there are an infinite number of such walks, but the strategy is predicated on purposeful walks (moving towards a goal). Examples of these derived movements can be seen in the experiments for the Multi-Team Multi-Agent simulation.

As a clarifying note, an intelligence model provides a strategy model to each team. The strategy model provides, via the assigned policies, behavioral models

for each of the agents. In this manner, it can be seen that these models provide movement and other actions for each of the agents under their control. As these agent follow their models, they are moving and taking actions within the partial view of the other team's agents. This means that another team can observe the results of the first team's model, but they cannot directly observe their model. This model must be inferred from the observed actions of the team being observed. The team implementing the strategy model is being directed along the Total MDD by their behaviors (each following their own MDD). As other teams observe these agents moving and acting along their MDDs, even when observed partially, they can begin to infer what those MDDs might look like. Likewise, they can begin to infer the behaviors that would generate such MDDs. Further, they can then begin to infer the strategy that would create such behaviors (through the policy assignments), and thus determine the most likely strategy that the observed team is following.

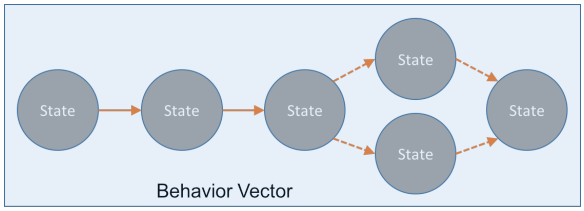
5.1.4 The Role of MDDs in Graph-Matching

Once these MDDs exist they can be used in two distinct ways, both interrelated. First, they are the models that the simulation uses at each layer of the simulation to progress the agents under consideration at that level. Second, importantly, they can be used to infer the intentions of other agents within that same level. Having these models (e.g., three different goal keeping policies for a soccer simulation) is mission critical to understanding which of these models the opponent team is following. These models are what enable the system to infer what the other agents are doing. To do

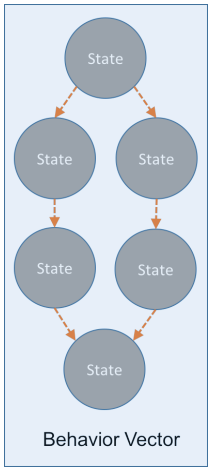
so, a matching is performed by comparing the observed actions of the agents being considered and matching those actions to these MDDs. Of course, an agent or team of agents is unlikely to observe every single action or transition being performed by the other team, so a belief network is built based on these models. As each observation is recorded, the percentage of belief in a particular model is modified. If the observed action fits within a model, the confidence (percentage belief from the belief network) is increased. If the observed action does not match, then there are two options: the system can eliminate a model from consideration, or it can ignore it. In the first case, the assumption is that a missing action from a behavior invalidates it from further consideration. The second case assumes a noisy system of imperfect observations and allows for a fuzzy classification of the most likely candidates. In either case, the end result is that the team is able to deduce the most likely model governing the actions of the other team (i.e., the policies they are following, and, subsequently their most likely strategy based on these policies). This process is detailed below.

Based on the hierarchical models provided in Chapter 4, the Strategy Inference Engine can take the provided strategy models and begin to match observations with them to form a belief network of the most likely strategy being used by a team that is being observed. The progression in Figure 5.19 shows how the Behavior Vector Models (2 examples are provided in this figure) are attached to the agent's FSA. Recall that the Behavior Vector is a subgraph of the MDD. It contains the movements assigned to a particular agent by their policy. This final figure offers the model diagram that shows the portion of the MDD that is covered by this particular strategy. If enough of

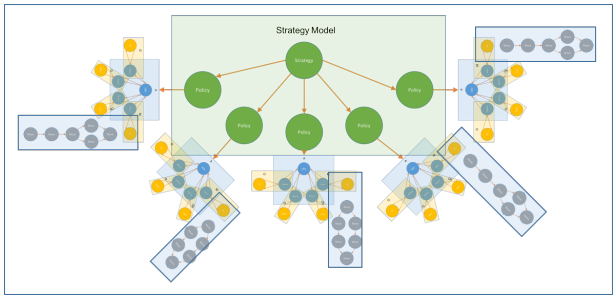
these movements are observed, it is reasonable to infer that the team being observed is following this strategy.



(a) Behavior Vector 1



(b) Behavior Vector 2



(c) Strategy Model with Behavior Vectors

Figure 5.19: Creating a Graph from the MDD for a Strategy

5.2 Using Graph-Matching to Infer Strategy in Multi-Agent Multi-Team Systems

For SiMAMT, the lowest level of consideration is the action. For each agent those are enumerated (e.g., move, cover, shoot, observe) by their behavior model. The behavior model informs the agent which actions to take (keeping the short-term goals in view) from a particular state. The behavior model is setup by the policy assigned to that agent by their strategy. The strategy model assigns the policies to each member of the team.

The behavior gives the progression / regression of moves by position according to their related probabilities as defined by the behavior's MDD. Each MDD contains a series of moves. Each move has a position with both a probability of choosing that position next and a speed with which the player will move to that location. For example, a move may say that a player will move to position 5 with a probability of 0.25 and a speed of 0.75 or position 6 with a probability of 0.75 and a speed of 0.5 (speed is expressed as a percentage of the maximum speed of the player). This is the probability of the moves based on the behavior, but this does not mean that the player will move, only that if they do this is the probability of them moving to either of these locations. The impetus to move is controlled by both the local policy settings and the overall strategy (detailed below). Again, we see here the individuality of the agent as viewed through the lens of the policy through the behavior because the speed of the move and probability of the move are both modified by the player's own speed and their willingness to move. These moves are chained together in probability clusters to form the bi-directional movement pathway that forms the core of each behavior. It is bi-directional so that each current move knows both the next moves and the

previous ones along the chain. With the information encoded in the behavior model and each policy having decision-making factors and a movement pathway, the policy can be incorporated into a strategy.

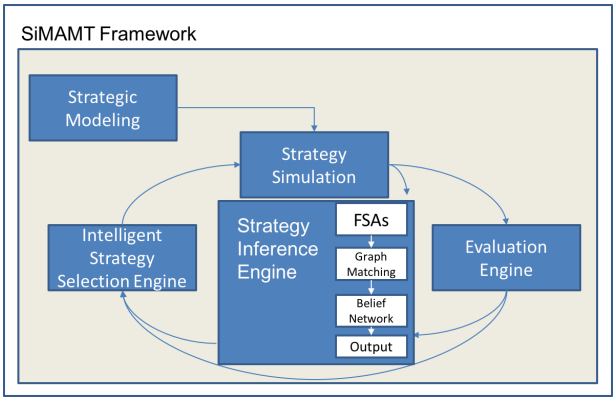


Figure 5.20: SiMAMT Framework: SIE

There are several working elements within the SIE, as shown in Figure 5.20. The first stage is comprised of finite state automaton (FSA) models that encapsulate the various strategies that the system is aware of (plus one more that is being built and modified in real-time to account for a strategy that the system has not seen - the $n+1$ model inherent in the system). The graph matching algorithm compares these FSA models with the data being generated by the simulation through the agent's observations to determine how well the observed actions match with the individual strategy models. Figure 5.21 shows a sample of these types of observations based on the Experiments scenario of 5-vs-5 Professional Speedball Paintball. The agent, denoted by the diamond shape, is located at position 306 looking around the right side of the obstacle towards position 509. The lighter colored enemy agents are observed (near position 514 and 310), while the other two enemy agents are not observed (near

location 309 and 510). In this instance, the agent would collect two observations, namely that there are two agents currently at those locations, and pass this along to the inference engine. It should be noted that the agent may or may not know which enemy agents these two are. They may be identifiable or not. In either case the inference engine can still makes it inferences, though if it can note which agents it is observing that helps the inference process. As a side note, as will be discussed in the Experiments section, the agents in this simulation are unlabeled.

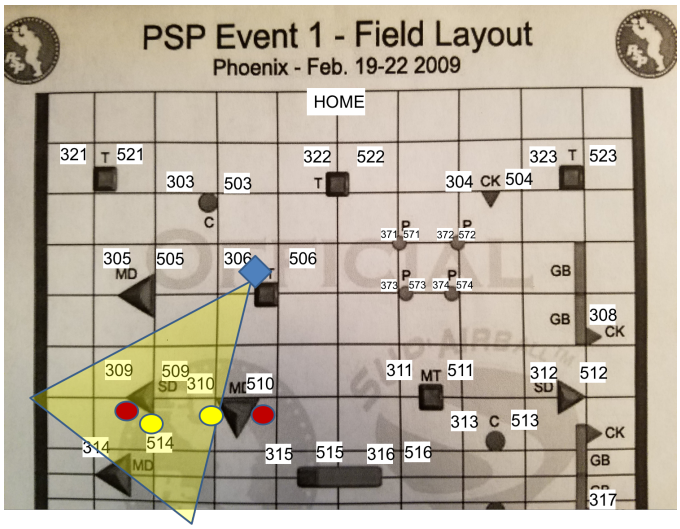
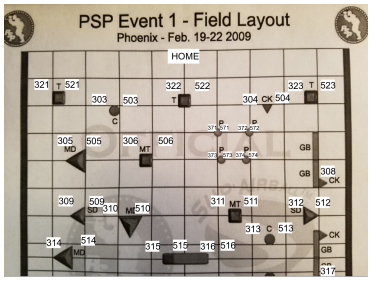


Figure 5.21: Agent Observation Example

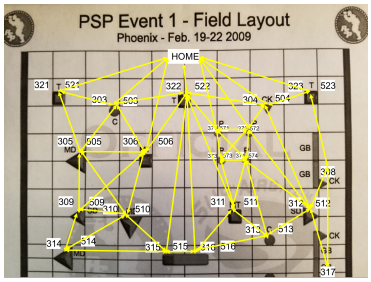
This data is collected into the belief network for final aggregation and analysis and forwarded from the belief network. The belief network is holding a single model for each known strategy, behavior, or any other element at that level. For example, if the current level is the strategy level and there are 4 known strategies then there will be 5 models in the belief network (1 for each of the known strategies and 1 more for the model being built as the simulation runs). As the matching progresses, as

detailed below, each node in the belief network constantly updates its belief for each model that that model is the best match to that one being used by the opposing team. The belief is essentially a percentage match of the observed elements with the known elements for each model.

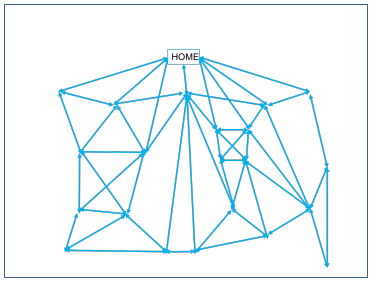
For illustration, the diagrams in Figure 5.22 show a sample strategy, σ_3, and the corresponding underlying behaviors (derived from each policy) that are being used by an opposing team that is utilizing this strategy. As the current team is making observations of the field they are noting the locations and movements of the players on the other team. With more and more observations there is more data for the SIE to use to update the belief network. In this example, the top figure shows that strategy with no observations. Each subsequent figure shows the progress of this node of the belief network as more observations are made. The result is the data shown in each subfigure where the beliefs of each individual behavior being observed are being aggregated into the total



(a) Speedball Paintball (PSP) Field



(b) Movements for PSP



(c) MDD: PSP

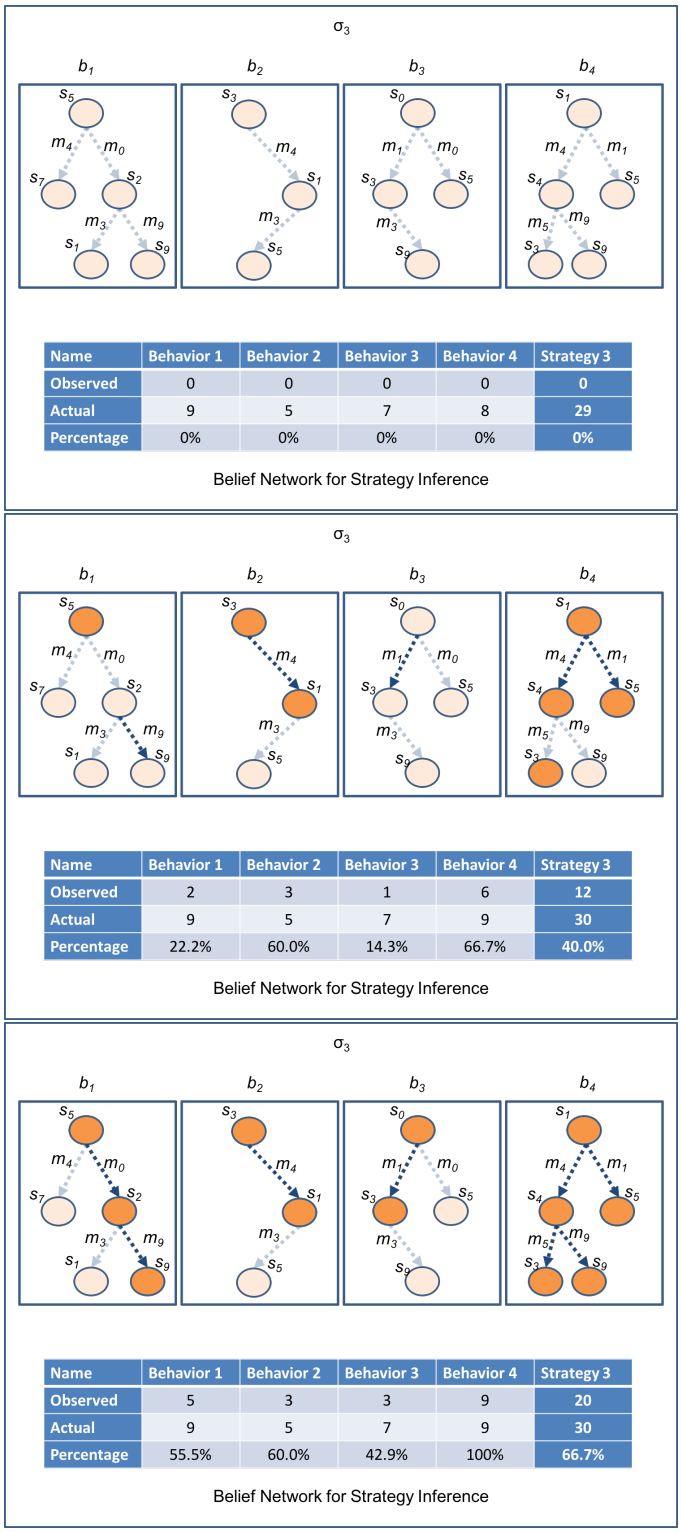
Figure 5.18: Professional Speedball Paintball MDD

belief that this particular strategy is the one being observed. This output is then sent to the Evaluation Engine for decision making and factoring.

5.3 Strategy Inference Methodology

5.3.1 Overview

In multi-agent strategic interactions, the complexity of strategic inference quickly leaves the computable range and becomes intractable. Learning in such an environment is even more difficult. When the strategies, and their associated policies, can be derived (or provided) probabilistically then they can be represented as either FSMs or PGMs. These models can then be encapsulated in two ways: first, as diverse sets of graphs for each such policy where the relevant walks in the graph represent strategic action chains (representing policies); second, as multiple isomorphic graphs where the weighting of the edges encodes the decision process. Both are described in the following sections after the background information is presented. This means that multiple agents interacting within the same environment can use strategies (with their related set of policies) to execute their actions and, thus, act intelligently. In this scenario, then, it is possible to reverse engineer this strategic interaction based on observations of the actions taken by a particular agent, the work of the SIE. By comparing the observed actions with the probable actions of each policy the belief network (BN) is formed that leads the particular agent to conclude the behavior of another agent within the system. The inference of the behavior leads backwards



Name	Behavior 1	Behavior 2	Behavior 3	Behavior 4	Strategy 3
Observed	0	0	0	0	0
Actual	9	5	7	8	29
Percentage	0%	0%	0%	0%	0%

Name	Behavior 1	Behavior 2	Behavior 3	Behavior 4	Strategy 3
Observed	2	3	1	6	12
Actual	9	5	7	9	30
Percentage	22.2%	60.0%	14.3%	66.7%	40.0%

Name	Behavior 1	Behavior 2	Behavior 3	Behavior 4	Strategy 3
Observed	5	3	3	9	20
Actual	9	5	7	9	30
Percentage	55.5%	60.0%	42.9%	100%	66.7%

Figure 5.22: Graph Matching for Belief Network

through to the policy. Combining the agent's observations of policies inferred from the observation of other agent's actions, the team leader can then infer the most likely strategy in play. In a system with increasing complexity, where calculating multiple factors may be time-prohibitive, the ability to match these candidate graphs (e.g., FSAs) with the currently forming belief network image (another graph) in real time offers an approximate solution.

5.3.2 Algorithm

To implement the strategy inference Algorithm 2 is implemented, shown here.

Algorithm 2 *StratInference* for Strategic Inference in Multi-agent Systems

```
1: Initialization
2: For i = 1, ..., NumTeams, Create a belief network for each other team
3: For i = 1, ..., N, Initialize each belief network (zero observations)
4: Main Inference loop
5: While not in an accepting (final) state
6:     For i = 1, ..., NumTeams, For each team do:
7:         For i = 1, ..., NumAgents, Aggregate each team member's observations
8:         For i = 1, ..., NumTeams, For each other team do:
9:             Set most likely strategy to the first known strategy
10:            Examine each observation and update each belief network
11:            If strategy belief > most likely strategy, update most likely strategy
12:    Update simulation state
13: Complete simulation
```

This algorithm will be described in terms of a multi-agent multi-team system, but it can be implemented in a single-agent system as well. The same process will work in simple one-on-one environments and in complex multi-agent environments. Examples of both will be provided in this research.

The algorithms starts by creating, for each team, a belief network associated with every other team. For example, in a three team scenario, team one would have a belief network for team two and team three. Each belief network is designed to hold the current most likely strategy that team is following. Continuing the example, team one would have one belief network for team two that is maintaining the current beliefs associated with team two. An example belief network is shown in Figure 5.23. The network includes beliefs about the strategy that is most likely in force for that team by aggregating the beliefs about the most likely policies in force for that team. These policy beliefs are based on beliefs about observed behaviors. This is explained in detail in the implementation section. The algorithm continues by initializing every element within the nodes of the belief network to zero, meaning that none of the elements have yet been observed. Detail on this process will follow.

The algorithm then shifts to the main inference loop. Because the Intelligent Strategy Selection Engine (ISSE) is a non-deterministic finite state transducer (NFST), it starts by processing input. This processing continues until the the system reaches an accepting state (i.e., a final state). In the case of a simulation, those accepting states would involve a circumstance that ends the simulation, like one team being eliminated, or a goal being reached. Next, every team then aggregates the observations of each team member into a single team observation. This is modeling a group communication system, but the system could also work without group communications (in that case, the individual agents would have their own belief networks rather than the team). Once these communications are aggregated,

they are processed. The processing steps through every observation, by team, and 'marks' each element within that belief network node that has been observed. The belief network is then updated based on these new observations and each layer of the belief network feeds forward it's beliefs. If the aggregated belief for a strategy is higher than the current highest belief then that strategy becomes the new most likely strategy in force for the team being observed. This continues for each team until that team has identified the most likely strategy for each other team, then the processing moves on to the next team.

Once completed, the simulation steps forward and the process continues until completion. This is the Strategy Inference portion of the SiMAMT framework, so it passes its output (the most likely strategy being followed by the other teams) to the ISSE for processing. The ISSE will then compare the team's current strategy with those of the other teams most likely strategies and make a decision on whether or not it is time for a strategy transition.

5.3.3 Application

Taking this belief network model and the strategy inference algorithm together, the system can begin to analyze the action of another team or agent and begin to reason about their most likely strategy. There are applications of the system in action provided next, along with experiments later in this chapter. The first example is a generic formulation, with examples, that apply (as modified, mentioned above, by moving from the theoretical to the general) to single-agent systems. This issue of

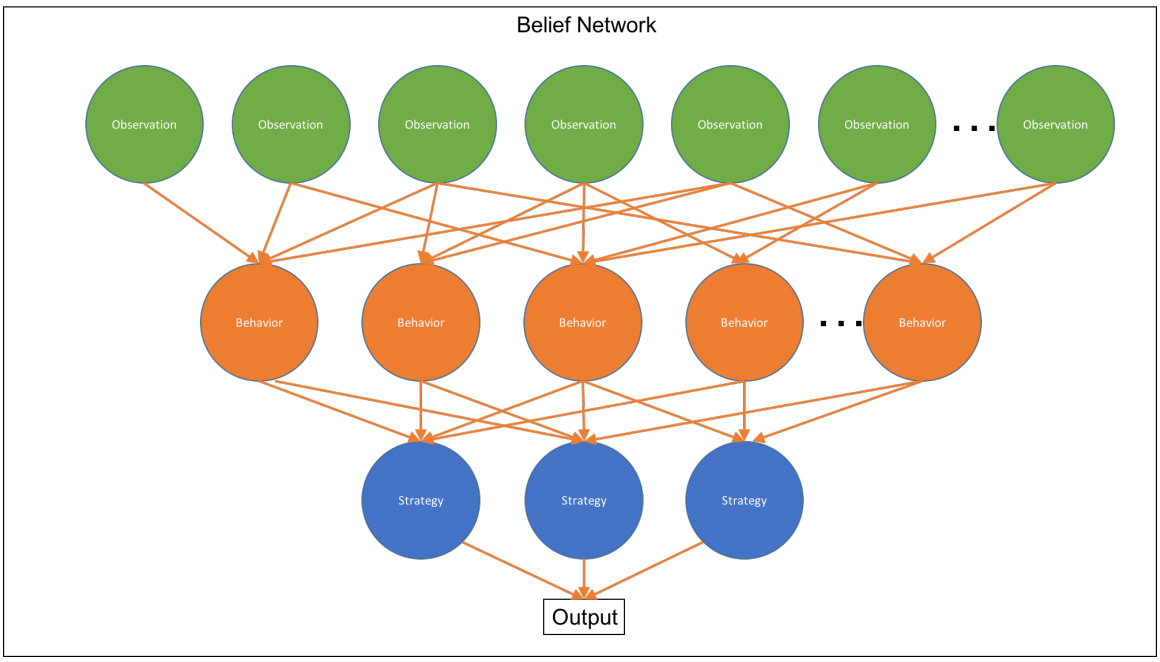


Figure 5.23: Belief Network

dealing with single-agent systems is the largest issue with the theoretical background material. The second example describes the multi-agent approach proposed by this research. This approach will work in both single-agent systems and multi-agent systems. As a result, the final proposed solution surpasses the first, single-agent only system, and can be used as a unified approach to both scenarios.

Example 1: Strategy Inference in Roshambo

In the Roshambo example the focus agent will start with a pre-determined strategy, σ_1, that selects its preferred policy, say 'LB3Rock', π_{LB3R}. Once play begins, the in force policy effectiveness will be measured by its behavior's performance and compared with standards (for instance, determining if it is beating the basic minimum probability baseline established by random game play samples). Each round the focus

agent's previous round is examined. The other agent's previous plays (and aggregate history) are examined and the finite state automata (FSAa) are updated. There is one model for each known policy, $\pi_{1..n}$ and one additional model that is being built from scratch to catch an unknown policy, π_{n+1}, in play. The policies are the models being examined and evaluated, but it is the observation of behaviors that informs this effort. Once these models are updated, the belief network is updated and the most likely candidate policy, π_c, is chosen. Once the focus agent has its best guess as to the policy the other player is using, π_c, it can determine the most likely move of the opponent. With this information, the focus agent can compare its own move choice with the most likely move choice of the opponent and determine if this move is still optimal or if a different move is needed. Additionally, if the move its own policy, π_1, is suggesting continues to be sub-optimal then the Evaluation Engine (EE) can recommend a policy change. The next policy can be selected from the subset of policies held in the strategy. Thus, the strategy σ_1 can shift to a better policy, π', as recommended by the EE. As play progresses, the goal is to have a the EE inform the strategy to select an optimal policy and stay ahead of the opponent by constantly shifting to better policies from within the subset of policies available to the strategy. Also, while observing the opponent's behaviors, it is possible to note when they change policies. As a result, the overlaying strategy can be inferred based on when and under what circumstances the policies shift. With this knowledge, the advantage shifts quickly to the focus agent as not only the moves of the policy can

be surmised but also the shifting of the policies and a greater understanding of the overall direction of the opponent's strategy.

Moving to the general scenario, this concept can be expanded to work in a team setting by having a strategy σ_{team} that incorporates team goals and coordinates team policies. Once the agents are coordinating their efforts as a team and competing with other teams, the next step is for a disadvantaged team to observe another, more successful, team's behavior and learn from it. In the sample game presented next, there is an objective periodic reward function that gives each team a score for its last round. Disadvantaged in this context simply means that there exists a team that is receiving a lower score due to its initial strategy than some other team. Strategy inference, then, is the process of this under-performing team observing a more successful team's behaviors, inferring its policies and strategies, and adapting its current strategy, adopting the other teams' strategy, or continuing with its current strategy. In this manner, each team can strive for improvement. This research proposes to use this evaluation and strategy shifting cycle as a method for increased performance for a disadvantaged team.

There is a list of possible strategies Σ for teams to utilize, and they vary in performance from optimal to minimal. The scenario is such that there will initially be one team that has an optimal strategy, σ^*, and another that has a lower performing strategy, σ_{min}. Through observation and pattern matching the under-performing team can determine the possible policies that the optimal team is utilizing and start adapting its own policies to match. Once these policies are known, and the shifting of

the policies is understood, then the strategy of the optimal team will be employed by the other team. Using a belief network, generated from a FSA model, that considers all possible strategies that could be being used by this optimal team, the under-performing team will enact the strategy, σ', with the highest probability of matching. Utilizing probabilistic transitions in this matching helps to model the noise inherent in observational data (Han et al., 2000). As additional behaviors are observed the belief network is updated to reflect the current state of the game and then adopting the most probable strategy again. In this manner, the under-performing team should approach the optimal strategy, σ^*.

It would also be possible to add a third team that is exhibiting an additional strategy that can compare imitation with adaptation. This third team would be working on cycling through strategies to find an optimal policy. The comparison would then be possible between the speed to reach optimality of imitation and adaptation. These factors may also be affected by team size, and that is another factor that will be tested.

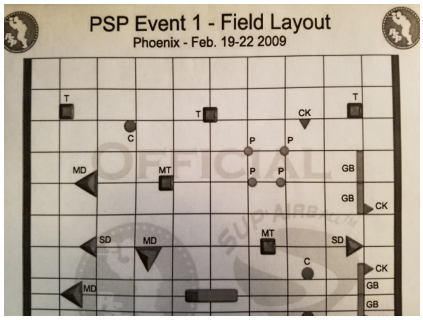
It is important to note that agent actions are continuous and not discrete, so recognizing discrete steps across these behaviors is important. This is recognized as a challenging problem (Han et al., 2000). The approach presented in this proposal utilizes an underlying FSA model with multiple entry and exit points to decrease this sensitivity, but it still requires the ability to recognize discrete steps. As noted before, the utilization of probabilities can decrease the noise of this discretization. Additionally, it should be clear that because this strategy inference is framed in such

a manner that its output (namely, the suspected strategy that the other team is using) is essential during the gameplay, this inference process must be online and interactive time. Further, any system that gave slightly more accurate results but did so after the game was over would clearly not be as useful as a best-guess inference engine that provided the insight when it was needed so that the focus team could modify its own strategy, σ_1 while it still mattered (i.e., while the game is still being played, or while the simulation is still running).

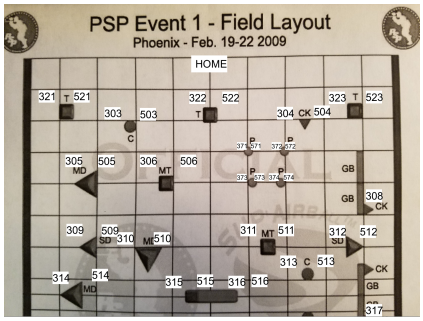
Example 2: Proposed Strategy Inference

In the more complex environments, the inference engine needs to build a belief network from the partial observations of the agents on opposing teams. As each behavior is being processed by the non-deterministic finite state transducer (NFST) it is processing an alphabet of input characters (for this research, a vector of probabilistic movements) and generating an alphabet of output characters (for this research, a vector of observations). Each agent is then being moved along the vector of movements probabilistically from position to position. At each position visited, the agent is in a location and has assumed a posture at that location. From this vantage point the agent can make two type of observations. The first is that the agent can make direct visual confirmation of another agent that is in their field of view. The second is that the agent can observe the transition of another agent (the movement of another agent).

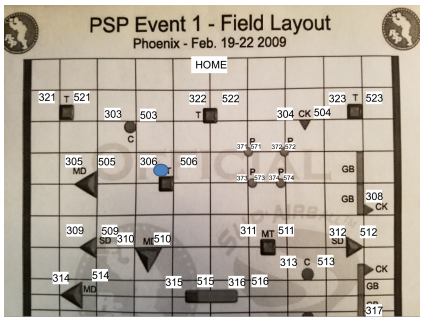
In the first type of observation the agent is looking from their own vantage point. The field of view of the agent is determined by the location and the posture. Figure 5.24a shows a particular field layout as an example scenario terrain (in this case, Professional Speedball Paintball). The locations within this sample field are then labeled in Figure 5.24b. An agent is then placed on the field in a certain position (location 306, posture is RightSide), represented in Figure 5.24c. The obstacles in view create a bounded frame of reference for the view plane of the agent. Figure 5.24d shows the view frustum for the agent from this position (recall that a position includes a posture or orientation). In this particular example, the agent can 'see' the agents within its view frustum (e.g., the green agents in location 310 and 514) but not those occluded by objects (e.g., the red agents in location 309 and 510). If another agent is in a position (that is, a location and a posture) where their reciprocal view plane is incident on the agent's view plane then the focus agent can record the observation. As may seem obvious, the other agent can also make the same observation. There are opportunities where this observation is not reciprocated, of course, such as when an agent is in a position where they can be observed, but their angle of view is directed elsewhere. In this figure, the enemy agent at location 310 can also see the agent, so it will make an observation to this effect. However, the enemy agent in location 510 can see the agent, but the agent cannot see it. The view frustums of the agents determine what they can and cannot observe. Additionally, Figure 5.25 shows the concept of a transition observation to demonstrate the second type of observation.



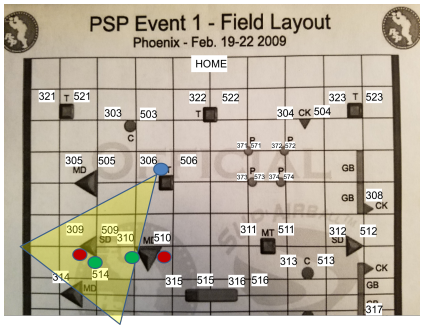
(a) Scenario Field



(b) Scenario Field Labeled



(c) Agent Placed on Field



(d) Agent View

Figure 5.24: View Frustum of Agent

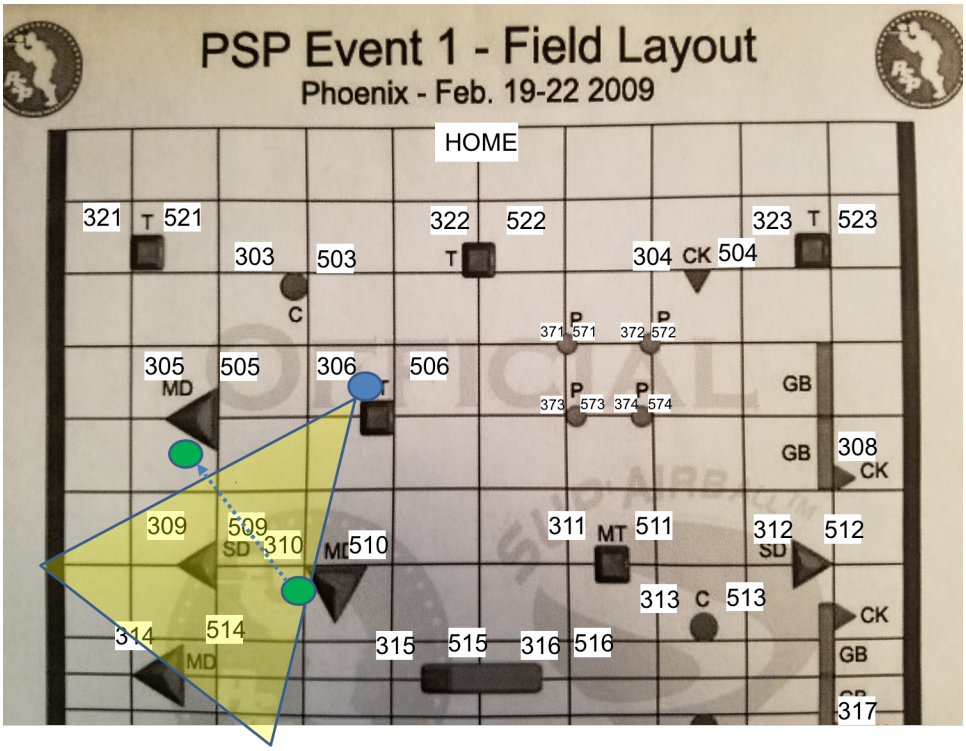


Figure 5.25: Agent View of Transition

The second type of observations, transitions, are similarly made. As an agent is looking at the field from their position they may observe agents performing their movements (recall that a movement is defined by a position, a speed, and a probability of making the move). When an agent makes a move they are traveling from one position to another at a certain speed (as prescribed by the movement). If the speed is slow enough then an agent looking in the direction of the movement can make a highly-qualified observation of the transition. These transitions are higher in weight than direct observations because they provide more information (in particular, two locations and a speed). In additions, when there enough observations made, there is

an additional probability of determining the probability of the observed transitions and thus knowing the full details of the movement.

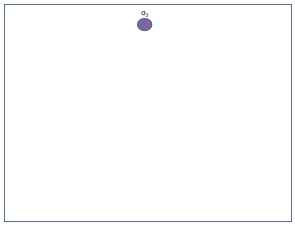
These observations are recorded in the output stream. Each record in the output identifies the agent who made the observation, the team to which they belong, and the observation itself. As a result, the teams can then aggregate these agent observations into one coherent view of the movement of the other team's agents throughout the field. Each observation is a piece of a hypothesis about which of the many strategies a team may be following. These individual hypotheses can be aggregated into policy graphs by taking each direct observation and making them a node on the graph. Any transitions that are observed represent the edges on these policy graphs. Each walk through the graph is a hypothesis about the behavior a particular agent is following.

Once the overall graph structure is complete then there exists a graph that represents a strategy. This graph can be combined with others as candidates for a belief network. In such a network each candidate graph is assigned a percentage that represents how much of the target graph is captured within a particular candidate graph. This process starts with building the strategy graph. Figure 5.26a shows the core of the graph, a strategy (e.g., σ_3). This strategy can then be expanded to reveal the mapping of the policies for each team member, shown in Figure 5.26b, and consolidated, shown in Figure 5.26c. Since each policy is a mapping from an agent (e.g., o_i) to a behavior (e.g., b_1), the graph can be expanded as a direct mapping to these behaviors, shown in Figure 5.26d, and consolidated, shown in Figure 5.26e. Next, this graph can be expanded further from these behaviors. Each behavior is

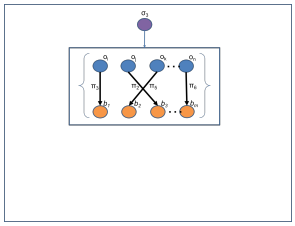
expanded to include its vector of movements and then consolidated by using direct mapping, shown in Figure 5.26f. This gives a representative graph that details and defines each strategy by its expanded children all the way down to the movement level. Now the matching can begin by comparing observations with known strategy models.

As observations are made movements can be determined based on locations observed and transitions witnessed. As these movements are aggregated they can be matched to the candidate graphs. This process is shown starting with Figure 5.27 where each unobserved movement is greyed out. The chart defines how strong the match is for each behavior and, correspondingly, for the particular strategy. Once observations are made the graph begins to be matched, as shown in Figure 5.28, where a number of matches have occurred. The updated belief network information shows the relatively low match based on these few observations. Finally, as the process continues, more observations are gathered and the resultant graph, Figure 5.29, shows the growing confidence that this may be the matching strategy. Of course, in practice, this would be only one of several candidate strategies being considered, and the belief network would return the match with the highest confidence.

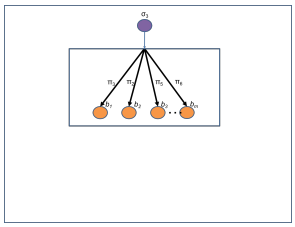
It is highly probable that there will be movements that are missed during the run of the simulation. As a result, the graph may be disjoint or unconnected. The graph matching algorithm detailed in this system is fault tolerant and accepting of partially observable data. As mentioned, this algorithm constructs a belief network that takes these observations as nodes and edges on a graph and attempts to match them with



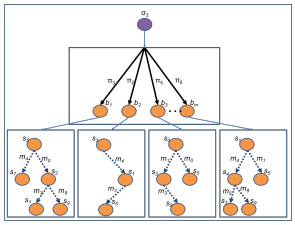
(a) Initial Strategy



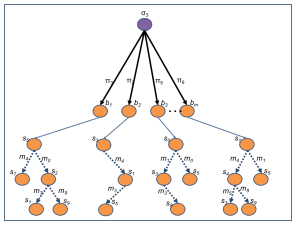
(b) Expand Strategy via Policies



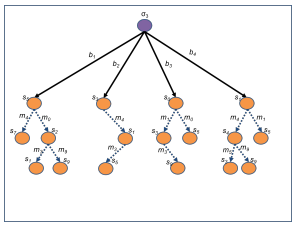
(c) Expand Policies via Behaviors



(d) Expand Behaviors via Movements

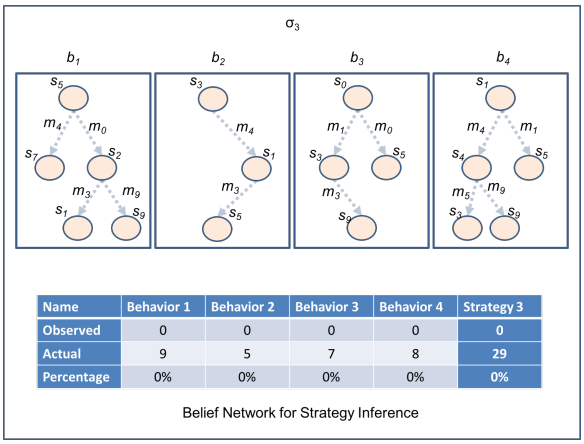


(e) Consolidate to Policies



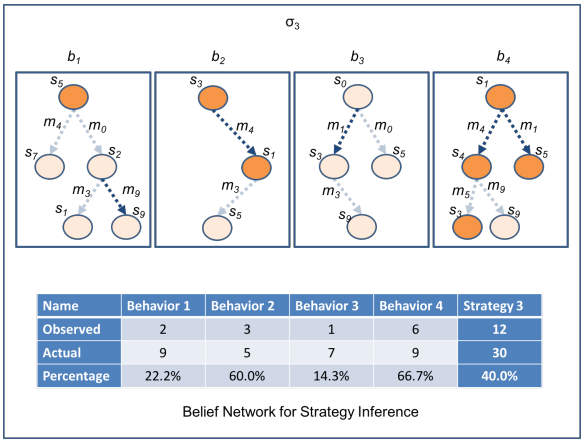
(f) Consolidate to Behaviors

Figure 5.26: Building a Strategy Graph



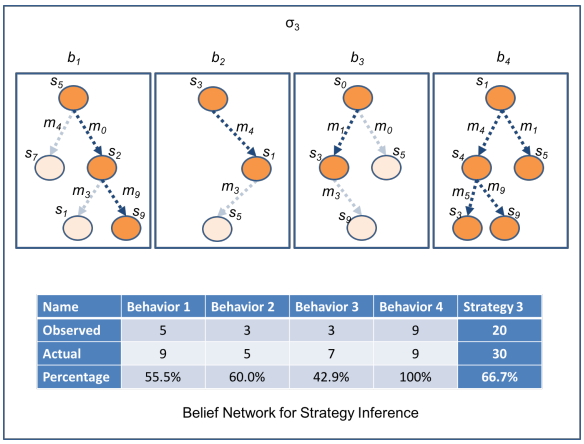
Name	Behavior 1	Behavior 2	Behavior 3	Behavior 4	Strategy 3
Observed	0	0	0	0	0
Actual	9	5	7	8	29
Percentage	0%	0%	0%	0%	0%

Figure 5.27: Graph Matching - Begin



Name	Behavior 1	Behavior 2	Behavior 3	Behavior 4	Strategy 3
Observed	2	3	1	6	12
Actual	9	5	7	9	30
Percentage	22.2%	60.0%	14.3%	66.7%	40.0%

Figure 5.28: Graph Matching - Initial



Name	Behavior 1	Behavior 2	Behavior 3	Behavior 4	Strategy 3
Observed	5	3	3	9	20
Actual	9	5	7	9	30
Percentage	55.5%	60.0%	42.9%	100%	66.7%

Figure 5.29: Graph Matching - Partial

known strategies. The belief network can be viewed as a percentage of likelihood that the current strategy being considered is the closest match to the actual strategy being followed. This is the generic formulation, but a specific application can be found in the Graph Matching section of the Experiments.

5.4 Implementation

Strategic planning and execution can be represented with multi-layered FSAs, as has been shown in Chapter 4, Section 4.2. In particular, the agent FSA will model the basic agent behaviors and their probabilistic progression. For example, consider the agent FSA shown in Figure 5.30 (shown earlier, but repeated here for clarity). At each decision interval, the agent model phase calculates a probability of making a move, seeking cover, and firing on the other team. They have a certainty of observation, both active (noting other agents in view) and passive (noticing zones from which they are receiving fire). These observations of the other agents positions, and most notably their transitions from position to position, are the key elements of the strategic inference. This starts with matching these observations (both active and passive) with the MDDs of the possible policies from which they could come. This forms a belief network wherein each node at this level of the network is a probable policy that receives votes from the confirmed matching of positions in observations with the various MDDs. While these observations add support for the belief network node, there are two methods where observations can remove support. When there is an observation of an agent at a position that is not part of the currently examined policy

it can either count as negative votes (reducing the overall belief of this policy being an in-force policy) or can be marked invalid (eliminating it from consideration). The simulation allows for both options, depending on the confidence of the observations. For example, if a transition from position A to position B passes position C, is the agent considered to have 'visited' position C? The simulation allows either method to work. Once these policies have been evaluated the belief network nodes are updated, and the belief network itself is updated by the aggregation of the previous nodes sending upwards their beliefs to the next layer of the network. This network is passed to the strategy analysis portion of the algorithm. This portion of the algorithm next evaluates each strategy by updating the belief nodes at the next highest level of the belief network where the nodes are probable strategies. There are also multiple methods to vote for strategies, just as there were with the policies. The result is a belief network that can then be passed to the SIE showing the most likely policies in force and the most likely strategies being implemented by the other team or teams.

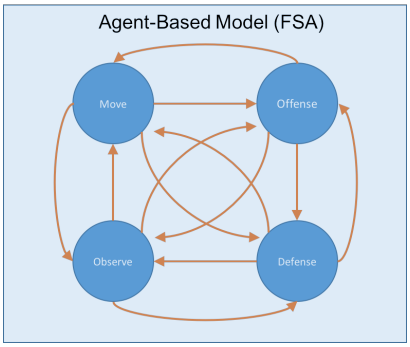


Figure 5.30: Agent Finite State Automaton Model

As an illustrative example, consider the game of soccer. There are a number of players per team (usually from 3 to 11, depending on the size of the field and the type

of game being played). Each player is generally assigned a position on the field (e.g., goalie, center-forward, right-wing). These positions can be thought of as policies, assigning a particular behavior to each player. This behavior can be realized as an MDD dictating the coverage area for the player (i.e., their assigned duties). In Figure 5.32, each of three players is assigned a behavior by their policy, and that policy is assigned to them by the team's strategy. This particular strategy is often called a fluid-zone defense where each player in the defense, because of the small size of team, is assigned a vertical strip of the field to cover from the goalie to mid-field. Figure 5.31 shows the field and Figure 5.32a shows the total MDD for the defense. In Figure 5.32b agent o_0 is assigned to the left flank defensive position and must cover that portion of the field. Similarly, Figures 5.32c and 5.32d show the assignments for agents o_1 and o_2, respectively. This leads to an aggregate movement graph that accurately reflects the movements of each player for a particular strategy (here, σ_0), as shown in Figure 5.32e. This graph can then be extracted and used as a model for this type of defensive play (Figure 5.33). This final figure can then be used as the graph for graph-matching based on similarly derived graphs for other agents and other teams of agents, even if only partially observed. As a result, the MDDs led to a mapping of behaviors, each of which is aggregated into a larger MDD that is a graph of the strategy.

Once such a graph can be built for each of the strategies known to the agent or their team, these graphs can form the library of graphs used for the graph-matching algorithm presented in this experiment.

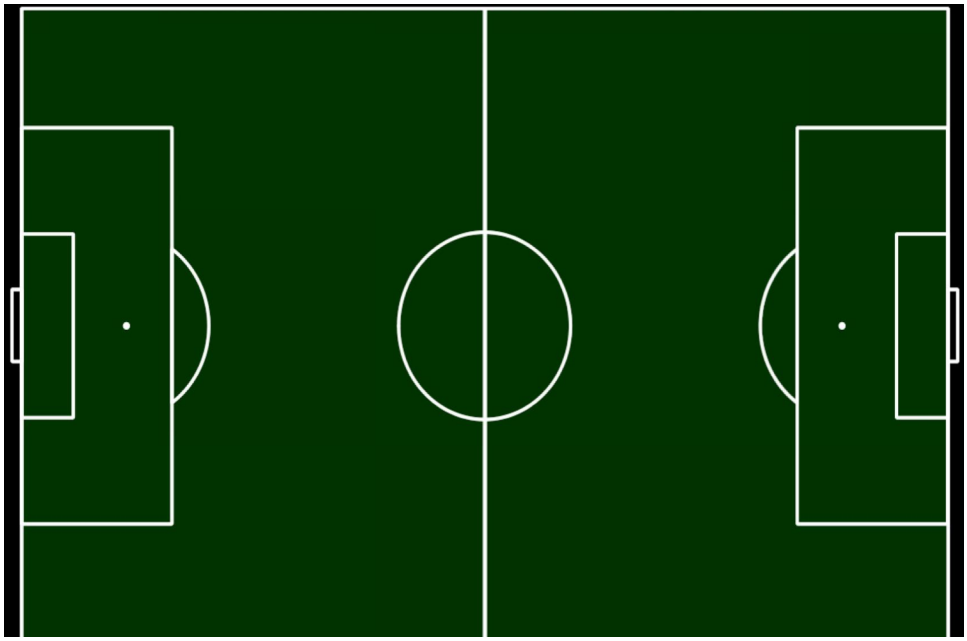
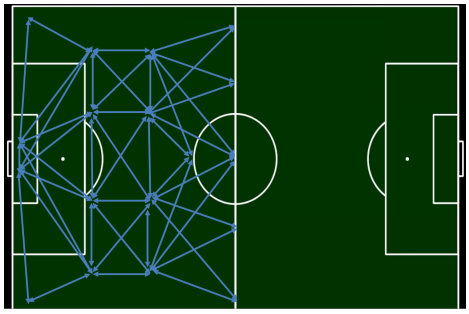


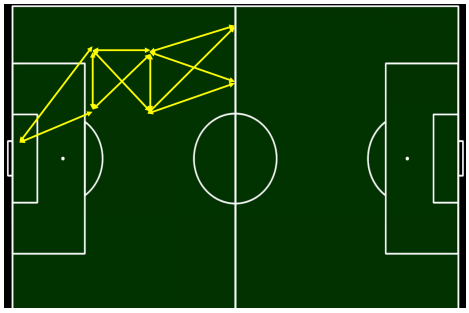
Figure 5.31: Official Soccer Field

Algorithms

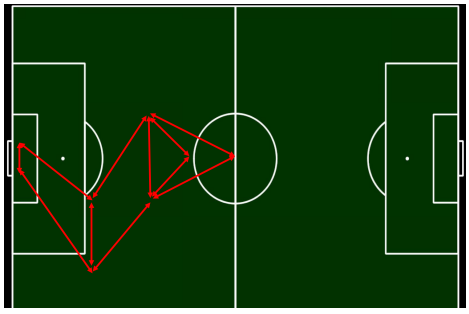
There are three algorithms that have been implemented in this work to perform the matching. These algorithm are in two parts, the calling code that is generalized for all three algorithms, and the matching function itself. The first part, shown in Algorithm 3, is the same for all three algorithms. It is the calling function that asks for the graph comparisons of the target graph with each of the candidate graphs and then returns the maximum match found. In essence, this function returns both the graph that matches most closely and the percentage of that match (i.e., the confidence). To implement the other two functions the only change would be the line that calls the function (it would just need to reflect the desired function name).



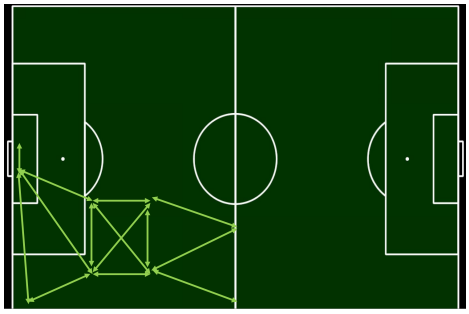
(a) Total MDD



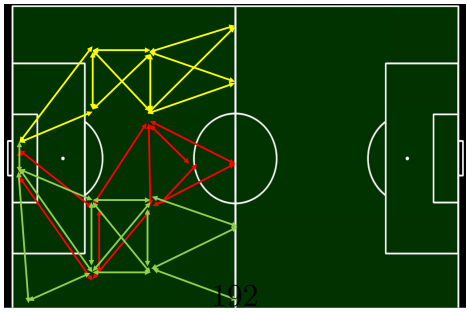
(b) MDD for Agent o_0



(c) MDD for Agent o_1



(d) MDD Agent o_2



(e) Graph of Strategy σ_0 (aggregate MDD)

Figure 5.32: Building a Strategy Graph from MDDs

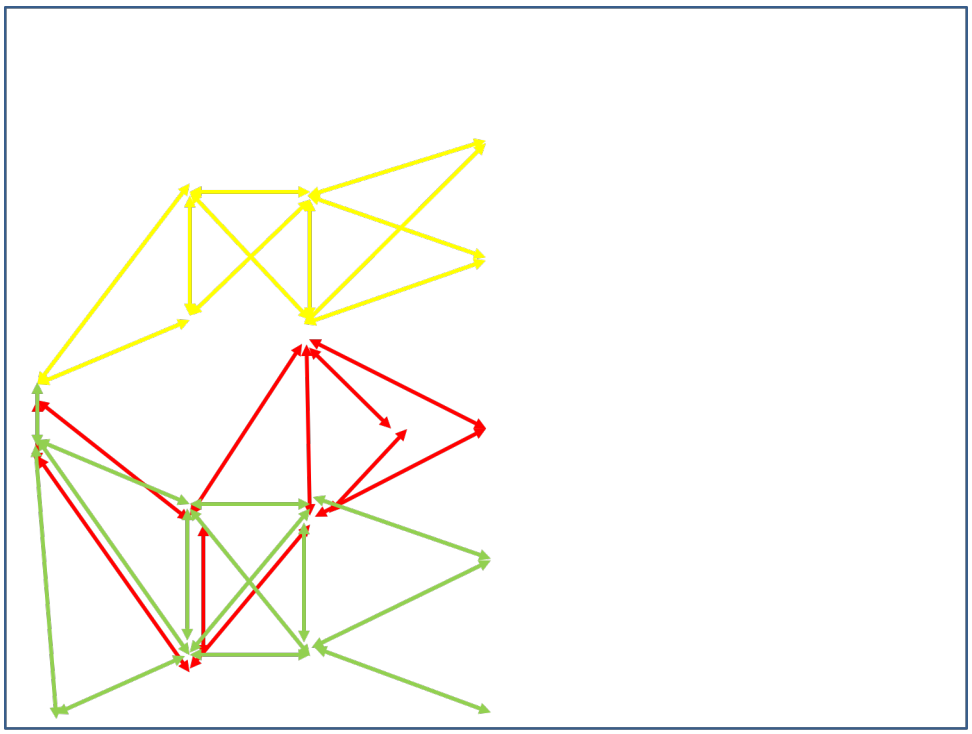


Figure 5.33: Graph of Strategy σ_0, Extracted

The first algorithm is the heuristic matching function that matches every vertex, edge, and connection, shown in Algorithm 4. A good reference for graph matching can be found in (?).

This function receives the two distinct graphs as input and produces a percentage match as output. This percentage match is designed to indicate how many of the vertices and edges match between the two graphs. In the case of isomorphism, the function would return a value of 1, or 100%. To perform the matching, the algorithm first sets two counters: *viewCounter* will increment for every vertex or edge viewed for comparison and also represents the total number of objects (i.e., vertices and edges) in the first graph; *foundCounter* will increment for every verified match (in this case, vertices with the same degree and upon whom are incident vertices of matching

Algorithm 3 *GraphMatch : Graph Matching of single graph with a set of graphs*

1: Input: $G_{targets} \subseteq G, g_{candidate}$
2: Output: $< g_{max}, \delta_{max} >$ (the best matched graph to the candidate and its confidence)
3: $\delta_{max} = -max$
4: $g_{max} = null$
5: **for all** $g_n \in G_{targets}$, $n = 0$ to $|G_{targets}|$ **do**
6: $m = \{null\}$ (matched graph elements)
7: $\delta_{g_n} = \text{Match}(g_n, g_{candidate}, m)$
8: **if** $\delta_{g_n} > \delta_{max}$ **then**
9: $g_{max} = g_n$
10: $\delta_{max} = \delta_{g_n}$
11: **end if**
12: **end for**
13: Return $< g_{max}, \delta_{max} >$

degree (without order specificity). The algorithm starts by looking at every vertex in the first graph and every edge incident on that vertex. For every such edge, the algorithm looks through the second graph to find a vertex of equal degree to the vertex. Once found, it looks to see if there is a connection for every edge from the first graph (i.e., edges with the vertices of the same degree) in this current vertex of the second graph. For example, if the first graph has a vertex with edges connecting to vertices with degrees of 2, 3, and 4, then the vertex under inspection in the second graph must also have edges that connect to vertices across edges with degrees that match. The order does not matter, but there must be the same number of the same degree vertices. Once completed, the algorithm marks the vertices and edges that it has used in m so that they are not used again in later comparisons.

In the case of graph subgraph isomorphism the inspection is complete when every vertex of the first graph is covered by similar vertices from the second graph. To

Algorithm 4 *HeuristicMatch : Heuristic Graph Matching g to g'*

```
1: FUNCTION: Match(g_n, g_candidate, m)
2: viewCounter, foundCounter = 0;
3: for all (V_i ∈ ν(g_n)) ∉ m do
4:    viewCounter++
5:    m += V_i
6:    for all (E_j ∈ E(V_i)) do
7:       viewCounter++
8:       for all (U_k ∈ ν(g_candidate)) ∉ m do
9:          if deg(V_i) == deg(U_k) then
10:            foundCounter++
11:            m += U_k
12:            for all (D_l ∈ E(U_k) ∉ m do
13:               if ((E_j ∈ m) ∨ (D_l ∈ m)) then
14:                  Break
15:               end if
16:               if deg((V_x ∈ ν(E_j) \ V_i) == deg((U_y ∈ ν(D_l) \ U_k) then
17:                  m += E_j
18:                  m += D_l
19:                  foundCounter++
20:               end if
21:            end for
22:         end if
23:      end for
24:   end for
25: end for
26: Return (foundCounter/viewCounter)
```

test for isomorphism, where the bijection is true, simply call the function again and reverse the graphs. If both return 1 then the graphs are isomorphic.

As stated, the return value is the percentage of elements found from the total number of elements.

The second algorithm is the initial approximation function that only focuses on matching vertices. This function is shown in Algorithm 5.

Algorithm 5 *ApproxMatch : Approximation Algorithm for Graph Matching g to g'*

1: FUNCTION: ApproxMatch(g_n, $g_{candidate}$, m)
2: Create conflict graph for g_n and $g_{candidate}$
3: $g_{conflict} = \{null, null\}$
4: **for all** $V_i \in \nu(g_n)$ **do**
5: $\quad \nu(g_{conflict}) \mathrel{+}= V_i$
6: **end for**
7: **for all** $U_j \in \nu(g_{candidate})$ **do**
8: $\quad \nu(g_{conflict}) \mathrel{+}= U_j$
9: **end for**
10: **for all** $V_i \in \nu(g_n)$ **do**
11: $\quad$ **for all** $U_j \in \nu(g_{candidate})$ **do**
12: $\quad\quad$ **if** $deg(V_i) == deg(U_j)$ **then**
13: $\quad\quad\quad$ **if** $\nexists E \in \mathcal{E}(g_{conflict})$ where $E = \{V_i, U_j\}$ **then**
14: $\quad\quad\quad\quad \mathcal{E}(g_{conflict}) \mathrel{+}= E\{V_i, U_j\}$
15: $\quad\quad\quad\quad m \mathrel{+}= V_i$
16: $\quad\quad\quad\quad m \mathrel{+}= U_j$
17: $\quad\quad\quad\quad$ Break
18: $\quad\quad\quad$ **end if**
19: $\quad\quad$ **end if**
20: $\quad$ **end for**
21: **end for**
22: Return $(|m|)/(|\nu(g_n)| + |\nu(g_{candidate})|)$

This function take in the two graphs as arguments for matching. It starts by creating a conflict graph to determine covering of both sets of vertices and edges. Normally it would be best to discover no conflict in such a graph, but we wish to find

a total conflict (this would indicate that all vertices and edges are covered). In this context, covering means that there is a vertex in the other graph that matches the current vertex in degree and in incident edges with vertices of similar degree (though here order does not matter). To begin, the conflict graph is empty. Each vertex from each of the two graphs is then added to the conflict graph. Once completed, the algorithm looks at each vertex in the first graph and compares it with each vertex in the candidate graph to see if there is one of similar degree. If so, is creates an edge (a covering) in the conflict graph to show this mapping. Both vertices are now added to the set of m so that each covered vertex is in this set (this will be used later to calculate the percentage of matching). At this point, since it has found a match, it breaks out of the inner loop and moves on the next vertex in the first graph. This continues until each vertex in the first graph has been checked for a match in the second graph. This approximation algorithm does not check each edge, though it could add that functionality within this loop much like the earlier Complete Matching algorithm. As this is an approximation algorithm designed to reduce run-time it is formulated without edge checking (other than stating that the vertices have the same number of edges incident on them since they have the same degree). This is an approximation of the subgraph isomorphism checking mention previously. To have it approximate isomorphism the same method could be used (i.e., call the function again and switch the first and second graphs).

This result of approximation is sufficient for the purpose of graph matching as used in this research as this research is trying to match the graph to a finite set of distinct

graphs. As such, the function returning a percentage match, even in approximation, suffices for determining the most likely match between the graphs. Further, since it is faster (clearly, since it is not checking each incident edge for neighboring vertices of similar degree), it is more desirable than higher accuracy that would take too long to calculate.

The third algorithm is the improved approximation algorithm, shown in Algorithm 6. This takes the initial approximation algorithm and improves it by quickly eliminating invalid candidates and performs in-line pruning to reduce the search space.

This improved algorithm is substantially the same as the previous one, but it adds a new selection process before it decided to do the larger matching. First, the algorithm compares the number of vertices in each graph. If they do not match in cardinality then there is no isomorphic match and the algorithm can exit with a quick 'no match'. If there are few vertices in the second graph than the first then there is no possibility of either a subgraph isomorphic or isomorphic match, so the algorithm can also exit in this case. Of course, if the second graph has more vertices than the first then there could still be a subgraph isomorphic match, so if that is desired then this check can be eliminated and the matching can continue. If the graphs pass this test then the algorithm proceeds. Next, the algorithm sorts the vertices of the graphs in descending order by their degree. The effect of this is that the list of vertices are now order monotonically decreasing from the highest degree vertex to the lowest. As a note, the overall complexity of the matching is much larger than that of the sorting, so even though there is a cost to the sorting, it is not relevant to the overall complexity

Algorithm 6 *ImprovedApproxMatch : Improved Approximation Algorithm for Graph Matching g to g'*

FUNCTION: ImprovedApproxMatch(g_n, $g_{candidate}$, m)
If unequal number of vertices, return fail
if $|\nu(g_n)| \neq |\nu(g_{candidate})|$ **then**
  Return 0
end if
Starting with highest degree vertex, if no match, return fail
Sort $\nu(g_n)$, $\nu(g_{candidate})$ descending by deg(V)
for all $V_i \in \nu(g_n)$ **do**
  for all $U_j \in \nu(g_{candidate})$ **do**
    if $deg(V_i) \neq deg(U_j)$ **then**
      Return 0
    end if
  end for
end for
Create conflict graph for g_n and $g_{candidate}$
$g_{conflict} = \{null, null\}$
for all $V_i \in \nu(g_n)$ **do**
  $\nu(g_{conflict}) \mathrel{+}= V_i$
end for
for all $U_j \in \nu(g_{candidate})$ **do**
  $\nu(g_{conflict}) \mathrel{+}= U_j$
end for
for all $V_i \in \nu(g_n)$ **do**
  for all $U_j \in \nu(g_{candidate})$ **do**
    if $deg(V_i) == deg(U_j)$ **then**
      if $\nexists E \in \mathcal{E}(g_{conflict})$ where $E = \{V_i, U_j\}$ **then**
        $\mathcal{E}(g_{conflict}) \mathrel{+}= E\{V_i, U_j\}$
        $m \mathrel{+}= V_i$
        $m \mathrel{+}= U_j$
        Break
      end if
    end if
  end for
end for
Return $(|m|)/(|\nu(g_n)| + |\nu(g_{candidate})|)$

analysis. The effect, however, is meaningful. The algorithm can now compare the first vertices of each graph to see if they match. If not, then there is no match for the graph and the algorithm can exit quickly. This can continue with each degree vertex. Essentially, if there is ever a mis-match then the algorithm can quickly return a 'no match' result, otherwise it continues. For clarity, it should be added that if the vertices do not match across both graphs then there is no match; however, the vertices matching in degree does not ensure a match. It is also important to note that starting from the highest degree vertex means that the graphs are being compared from there most 'fragile' element (that which is least likely to match).

Finally, the algorithm continues just as the previous algorithm without any changes. This improved approximation algorithm works as well as the previous approximation algorithm but has a higher chance of exiting the process earlier. The complexity remains the same, as stated earlier, though in the average case it will perform just as well on accuracy but better on time.

These algorithms are used in the Experiments detailed at the end of this chapter and their relative performance is evaluated. With the algorithm in place to match the graphs, the belief networks can be built and evaluated. The belief network will use the results of the graph matching to determine the percentage of matching for each of the graphs and report this as the belief, or confidence, for the match.

The output from these algorithms is a percentage match. This is one method of forming the belief network that is core to the framework and the simulation. The purpose of the the belief network is to track changes over time and update

beliefs accordingly. In this manner, the framework is maintaining a matching for each candidate graph. While the algorithms mentioned previously can be used to compare a single graph to a set of graphs and determine the best possible match, it is more useful in practice to maintain these values for each of the graphs. To this end, the individual functions can be called independently as needed to determine these values. This data is collected into the belief network for the framework.

The belief network is a flexible object that can be defined in multiple ways. The SiMAMT framework will work with any belief aggregation that reports the best available match (i.e., this highest belief) among several graphs or models. There are many options available, but an example belief network was shown earlier in Figure 5.23. It shows the observations at the highest level. These observations are the actual position or movements of the players on the field or other agents within the simulation. Each of these observations are part of a behavior, or perhaps several different behaviors. When the observations are viewed collectively, they can provide insight into probable behaviors that are being observed. At the observation level the belief network nodes are providing a belief that they have viewed that particular observation. In a partially observable environment, for example, there may be 70% belief that the observed movement actually occurred. In any case, these observations are linked to the behaviors that have these particular movements or positions as a part of them. As a result, at the next level of the belief network, the next layer of nodes aggregate the observation beliefs that link to them. The nodes at this level represent behaviors. Once summed, these aggregate beliefs represent the systems'

belief that this particular behavior is being observed. In like manner, the individual behaviors are linked to the strategies that implement them (indirectly, as mentioned before, because the strategy assigns an agent to a policy, and the policy assigns the behavior to the agent, but this is compressed here for simplicity). The aggregated beliefs of each behavior reflects in the overall belief that this strategy is the one being observed. Finally, the belief network outputs the strategy that has the highest belief. This output is the most likely strategy being followed based on the fully aggregated values (passed up each layer of the network). The observations represent the most factual elements of the belief network, though they may have only been partially observed. As the results are passed to each layer of the network there is an increasing distance from the actual facts of the observation, but they still hold valuable information in summary. While the error of the observations may increase with each layer of the network, the matching does not require a 100% match, just a clarity from other samples in the matching. In other words, as long as the belief network can discern the differences in samples (e.g., among three different strategies) it is effective.

As stated, there are many formulations for belief networks, but this one is strongly representational. It is also the model of belief network used in this research. Each node is managing belief, and that mechanism is described next.

Loopy Belief Propagation is again used for this belief network. It is repeated here so that this example can stand on its own without the reader having to reference back so far in the research. We chose to use loopy belief propagation (Pearl, 1988).

This formulates the increases likelihood of candidates based on a message-passage paradigm wherein each observation sends messages to each hypothesis until either a time limit is met or a criterion is satisfied. In Equation 5.1 we see the messages passed from the variables to the factor, or for our purposes, the observations passed to the policy candidates. Next, we aggregate these messages around each hypothesis (candidate policy), as shown in Equation 5.2. This process is then repeated at the next hierarchical grouping (i.e., the policies are now the observations and the strategies are the candidates).

$$\forall x_v \in Dom(v), \mu_{v \to a}(x_v) = \prod_{a^* \in N(v) \setminus \{a\}} \mu_{a^* \to v}(x_v) \tag{5.1}$$

$$\forall x_v \in Dom(v), \mu_{a \to v}(x_v) = \sum_{x'_a : x'_v = x_v} f_a(x'_{v^*}) \prod_{v^* \in N(a) \setminus \{v\}} \mu_{v^* \to a}(x'_{v^*}) \tag{5.2}$$

This leads to the convergence of these beliefs, for the variables, Equation 5.3, and for the factors, Equation 5.4. This convergence gives us the result of the likelihood of candidate policies within the system, and, thus, the candidate strategies that are comprised of these most likely policies. Granted, this is a double probability, and, thus, a double inference, but it is the method or hierarchical aggregation that best infers the strategy in-force.

$$p_{x_v}(x_v) \propto \prod_{a \in N(v)} \mu_{a \to v}(x_v) \tag{5.3}$$

$$p_{x_a}(x_a) \propto f_a(x_a) \prod_{v \in N(a)} \mu_{v \to a}(x_v) \tag{5.4}$$

The SIE takes these aggregate belief networks and evaluates the performance of the current strategy, the performance of the candidate strategies (most likely strategies in place based on the belief network), and the performance of optional strategies. Once this data is available, the inference engine recommends the most likely next strategy to implement based on these comparisons. If the current strategy is best, or within ϵ of the the best, then no change is made. If not, the best optional strategy is selected (either a better performing strategy from the evaluation or the best counter-strategy to the one suspected that the other team is following) and put in place. This means that the players are all reassigned roles (assigned by the policies), policies (their movement through the field), and their decision making factors. The team instantly begins to realign itself with the new strategy and the process starts over again (though with more experience). The experiments below show the results of the simulation (the veracity, complexity, and expressibility), the ability to implement and follow strategies (hierarchical policy networks), and the ability to infer opponent strategy and switch strategies in interactive time (the SIE).

Once these strategies have been evaluated the belief network is passed to the evaluation engine portion of the framework.

5.5.1 Introduction

Building on the example offered earlier of the game of Roshambo and RPSLS, the larger claim of strategy inference and strategy imitation needed to be tested. To this end, an inferior strategy was chosen for the target player and a superior strategy given to the other player. This set up the target player to lose initially. The target player should then cycle through its policies within its strategy to select the best performing policy. This is the first result that was desired to be shown, that the player could find the optimal policy from within its current strategy. Next, the goal was expanded to allow for a meta-strategy that could choose from among the available strategies which strategy will perform best. This is borne out in the experiments below.

5.5.2 Inferring Strategy in Multi-Agent Systems

To infer the most likely strategy that the opponent is using, the agent examines each action taken by that opponent. By creating a mapping of possible moves predicted by each strategy, and the actual move taken by the opponent, a belief network is created. The nodes in this network accumulate votes as the game progresses. In this context, a vote is a match between the predicted move for a given strategy and the actual move taken by the opponent, that is, each model produces a corresponding graph representing it and then returns the next (most likely) move along this graph

based on the model. If the predicted moves on the graph match the actual move of the observed graph progression, a match is counted. As play continues the most likely strategy emerges. This tally can then be used to select a proper counter-strategy for the most likely opponent strategy in place. The table, 5.1, shows the various trials of the experiment with the prediction accuracy.

Opponent Strat	Initial Player Strat	Recognition Steps 1	2	3	Avg
RockBias	ScissorBias	18	22	12	**17.3**
ScissorBias	PaperBias	9	8	9	**8.3**
PaperBias	RockBias	15	6	19	**13.3**
	Average	**14**	**12**	**13.3**	**12.97**

Table 5.1: Strategy Inference Results

The table, 5.1, shows the initial strategies for the opponent and the player. These initial strategies were chosen in a biased fashion to create a disadvantage for the Strategy Inference engine. The third, fourth, and fifth column show the total number of rounds that passed before the Strategy Inference engine correctly recognized the strategy the opponent was using. In those cases where the correct pattern was recognized early on, but then the engine shifted strategies away from the correct strategy only to shift back to the correct one, the latter was used (i.e., the higher number, or the last time the correct strategy was chosen). There were three trials done with each of the selected disadvantaged strategies (shown in each column) and then these were averaged across strategies and across trials. The overall average is 12.97 rounds. This means that in less than 13 rounds, on average, the agent was able to correctly select the opponents strategy. This confirmed that the Strategy

Inference engine was able to correctly select the strategy in play from the complete list of available strategies ($n = 21$). This is not the same as determining a strategy in general without any foreknowledge, admittedly, but it is foundational to the concept of Strategy Inference.

5.5.3 Implementation

To see how this new information, the ability to determine the opponent's strategy through inference, would impact gameplay, further experimentation was implemented. This further experimentation added a strategy selection element into the code. Now the agent can not only infer the most likely (candidate) strategy of its opponent, it can change its policy to counter such a strategy. The agent now references its strategy guide (i.e., its listing of policies within its strategy) to select the best counter-strategy to the candidate policy. To elaborate, what is observed is an action. This action is part of a behavior (in a single-agent system, this is the same as the policy). The inference of the strategy, then, is based on knowing the list of policies within a given strategy. There could be such a policy that it is included in multiple strategies, so in that case the Strategy Inference engine would have to observe the shifting of policies within the game to understand which strategy was actually in place. For now, the experiment was made with mutually exclusive policy lists within each strategy so that recognizing a policy was sufficient to identify its strategy. This concept will be tested further in the final experiment, but for now we wished to show that the inference process could inform the strategy-based system to select the best counter-policy to

overcome the advantage of the opponent. The results of this experimentation are show in Table 5.2.

Opp Strategy	Initial Player Strategy	Without Inference			With Inference		
		Opp	Player	Ties	Opp	Player	Ties
RockBias	ScissorBias	4844	3415	1741	2584	6109	1307
RockBias	ScissorBias	4808	3466	1726	2595	6002	1403
RockBias	ScissorBias	4815	3444	1741	2525	6121	1354
	Average	**4822.3**	**3441.7**	**1736**	**2568**	**6077.3**	**1354.7**

Table 5.2: Counter Strategy Results

5.5.4 Experimental Results

These results show that the improvement in overall performance is significant. There is a clear advantage to inferring the opponent's strategy through policy observation and selecting an appropriate counter-policy from the agent's strategy. In this experiment the win rate (percentage) went from 34.4% without inference to 60.7% with inference. As noted before, another important characterization of this game is the *not lose* percentage, which jumps from 51.7% without inference to 74.2% with inference. As an additional note, one weakness with the original formulation of policy and strategy implementation was that any strategy (where a player has some dominant play) loses to the baseline random strategy. It is important to note that the experiments with inference will actually break even with the random strategy. This means that the inference can even help in the case where the strategy is purely random without significant penalty. The agent, of course, realizes that the best

counter strategy to a random strategy is to also play randomly. With the recognition of the strategies occurring so quickly there is very little variation in the scores.

5.5.5 Conclusions

While it is not surprising that choosing a better policy produces a better result, recall that the point of the experiment was to prove that it is possible to infer the strategy of the opponent through graph matching and thus counter it with interactive time analysis during play. This speed at which this recognition takes place, as shown in Table 5.1, means that it is entirely possible to recognize the strategy in play and react to it in interactive time. This result proves that, within this limited player interaction, and within this limited strategy set, inference is credible and counter-strategy selection practical.

To take this to the next level, with a much more complex interaction, the research focus shifted to a more fully-realized game.

5.6 Experiment 2: Approximation Algorithms for Homeomorphic and Isomorphic Probabilistic Interactive Time Graph Matching

For strategy inference to work in larger systems, given the complexity of the models, there must be a method of matching these models that can be performed in interactive time. To that end, the approximation algorithm for interactive time graph matching was developed. This plays a large role in the SIE to perform the matching that forms the complete method of full-matching (one application of the SIE where there

must be definitive results) and the partial method for belief networks (where the value returned is a percentage of matching in order to perform in partially-observable non-deterministic systems). The work, in its entirety, is presented here.

In many common gaming and real-world scenarios agents are trying to predict the behavior of the other agents. This assumes that there is some underlying strategy that these players are following, that such strategies can be inferred, and that a reasonable player can counter such strategies in interactive time. These strategies can be modeled as various structures such as graphs, finite state automatons (FSAs), or probabilistic graphical models (PGMs). With these models created, one approach to best determine which strategy an agent is following is to match prospective graphs, built from observed behaviors or policies, with known graphs, representing previously learned strategies and policies. While matching two graphs can be done in super-linear time in the small scale, the matching problem quickly becomes NP for the more complicated cases. This leads to a well-known NP-Complete problem (e.g., (Kumar et al., 2011)) when one considers homeomorphic graphs (one is a subgraph of the other) and isomorphic graphs (the bijection is true (i.e., homeomorphic in both directions)) and their matching. Isomorphic graph matching is much more complex (Vazirani, 1989). At scale, most solutions for graph matching utilize highly-parallel processes running on high-performance computing clusters. This is intractable for interactive time low-power computer systems. This research presents an approximation algorithm for graph matching in order to accomplish strategy inference in interactive time multi-agent systems.

5.6.1 Introduction

It has been shown in this research so far that strategies offer significant performance enhancement to artificially intelligent agents, that strategies can be recognized in interactive time when complexity is limited, and that AI agents utilizing strategy inference will outperform their originally superior opponents in the experiments detailed herein. To do strategy inference in more complex environments necessitates discovering better approximation algorithms for this kind of graph matching.

Classical machine learning requires repetitive trials and numerous iterations to begin to form a hypothesis as to the intended actions of a given agent. There are numerous methodologies employed in an attempt to reduce the number of examples needed to form a meaningful hypothesis. The challenge arises from the difficulty created by the diversity of possible scenarios in which the machine learning algorithm is placed. Given enough time and stability a machine learning algorithm can learn reasonably well in a fixed environment, but this does not replicate the real world very accurately. As a result, we utilize strategies, defined and explained in the previous sections. Strategies offer an opportunity to encapsulate much of this policy-space into a compact representation. They have to be learned as well, but they are transmutable to another instance of a similar problem. Additionally, they can be pre-built and then modified to suit the exact situation. If these strategies are represented as graphs then they can be classified, categorized, identified, and matched. In particular, they can be represented as a variety of highly expressive graphs such as

PGMs, FSMs with probabilistic progression, or other graph structures composed of complex elements. The strategy inference engine uses FSAs to build a belief network of candidate strategies from which it selects the most likely strategy an opposing agent is using. This matching is workable in fixed-size implementations and low-complexity environments. It is also restricted to known strategies so it does not learn new strategies currently nor does it approximate solutions. The belief networks utilized are successful in estimating solutions, but they require more experience than may be available in a real-world scenario. To learn strategies and form hypotheses in interactive time we utilize approximation algorithms. This research introduces an approximate solution to interactive time complex graph matching. First, we explain and propose algorithms for an exact solution. Next, an approximation algorithm is introduced to create approximate solutions in interactive time. After this, the approximate solution is improved upon to produce the final approximation algorithm. Finally, there is an analysis of the solutions and relative performance of each algorithm in a variety of applications.

5.6.2 Implementation of Graph Matching

Graph matching is complex enough on its own (e.g., (Kumar et al., 2011), (Vazirani, 1989)). Adding the subgraph generalization of homeomorphism makes this substantially more difficult and more complex in both time and memory. Further, the bijection of isomorphic matching adds even more complexity and an increased memory requirement. Because of this complexity, there needs to be a method of

approximating solutions to such matchings. This may include both approximate culling, where the target set of graphs or graphs may be reduced through simple algorithmic considerations, and approximate matching, where the partial candidate graph is best-matched with target graphs that remain after culling.

In multi-agent strategic interactions, the complexity of strategic inference quickly leaves the computable range and becomes intractable. Learning in such an environment is even more difficult, as previously shown. When the strategies, and their associated policies, can be derived (or provided) probabilistically then they can be represented as either FSAs or PGMs. These models can then be encapsulated in two ways: first, as diverse sets of graphs for each such policy where the relevant walks in the graph represent action chains (representing policies); second, as multiple isomorphic graphs where the weighting of the edges encodes the decision process. Both are described in the following sections after the background information is presented. This means that multiple agents interacting within the same environment can use strategies (with their related set of policies) to execute their actions and, thus, act intelligently. In this scenario, then, it is possible to reverse engineer this strategic interaction based on observations of the actions taken by a particular agent. By comparing the observed actions with the probable actions of each strategic possibility, a belief network (BN) can be formed that leads the particular agent to infer the strategy of another agent within the system. This requires the ability to match these candidate graphs (e.g., FSAs) with the current best-match graph from the belief network in interactive time.

To accomplish this task, this research seeks to first implement a complete matching (exact graph match with all vertices and edges identical) solution, as informed by the various reference papers and this research. This complete matching is then optimized to perform complete matching across graphs and to consider subgraph isomorphs and isomorphs. Next, an approximation algorithm is presented to both cull the target set and to provide approximate matching in interactive time, thus realizing the goal of recognizing strategies in real time. After these algorithms are fully implemented, several experiments are conducted to validate the hypothesis proposed herein. To be considered a success, these algorithms must find matchings and must do so in less time, less memory, or both. The methods are analyzed and compared according to accuracy, complexity and time.

Homeomorphism and Isomorphism

Two graphs which contain the same graph vertices connected in the same way, perhaps with additional vertices, are said to be homeomorphic. Formally, two graphs G and H with graph vertices $V_n = (1, 2, ..., n)$ are said to be homeomorphic if there is a permutation p of V_n such that the resultant graph is a subgraph of the former. Figure 5.34 provides an example from (Sepp, 2015).

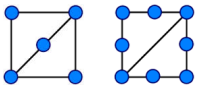


Figure 5.34: Homeomorphic Graphs

Two graphs which contain the same number of graph vertices connected in the same way are said to be isomorphic. Formally, two graphs G and H with graph vertices $V_n = (1, 2, ..., n)$ are said to be isomorphic if there is a permutation p of V_n such that (u, v) is in the set of graph edges $E(G) \iff (p(u), p(v))$ is in the set of graph edges $E(H)$. Figure 5.35 provides an example from (Windsor, 2015).

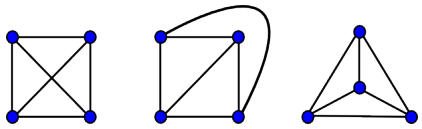


Figure 5.35: Isomorphic Graphs

The cost of calculating isomorphism and subgraph isomorphism (or any variant of homeomorphism) is high. There may be graphs that are homeomorphic and not isomorphic, but there are no reasonable graphs that are isomorphic but not homeomorphic. As a rule, we wish to consider isomorphism and subgraph isomorphism as we are not dealing with graphs that can morph. These intuitions will be used later in the intelligent pruning section of the approximation algorithm to cull the target list.

A complete matching would be one where all vertices and edges appear in both graphs (ensuring that the two graphs are identical because their lists of vertices and edges are identical) and every vertex of matching degree is connected to the same number and degree of vertices in both graphs. Figure 5.36 shows the complete match between the candidate graph on the right-hand side of the bar and its ideal target from the target graphs on the left-hand side of the bar. While this is ideal, the cost to

calculate it is too high. Instead, we wish to approximate the performance of complete matching with the approximate matching algorithm proposed herein.

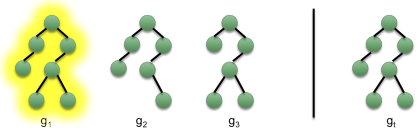


Figure 5.36: Complete Matching

Strategy Representation

Strategies can be represented as a FSA. For example, consider the simple game of Rock-Paper-Scissors (Roshambo). In Figure 5.37 the game is represented as a FSA. As an example of how this model would look once the strategy has assigned a policy to it and the behavior has been applied, Figure 5.38 shows the strategy *RockBias*. In this example, the play 'Rock' is twice as likely as any other move. As a result, the transitions to *Rock* are 0.50, while all other transitions are only 0.25. The FSA can continuously produce next states for each turn where a play is required.

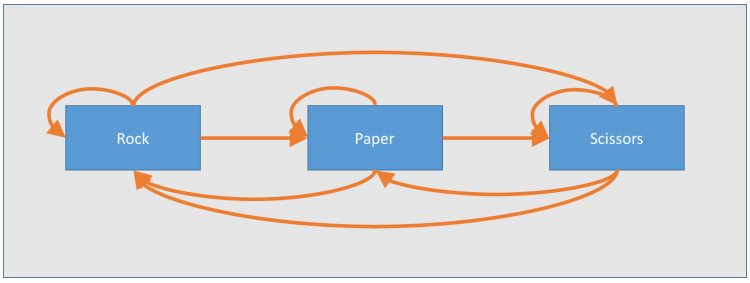


Figure 5.37: RPS FSA Model

In this assembly, the graph is isomorphic for all players, but the weights of the edges vary (i.e., the policy is encapsulated in the transitional probabilities of each

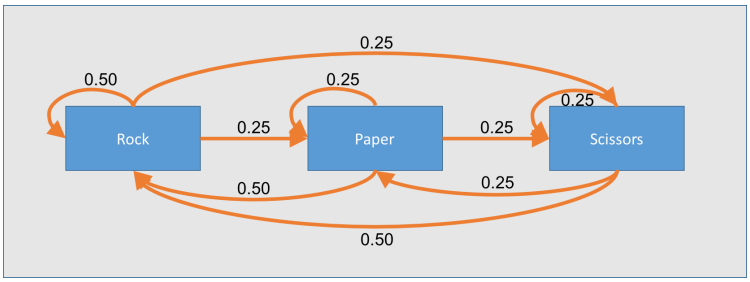


Figure 5.38: RPS FSA Model - Rock Bias

edge). While there are many such sample graphs that could be considered, there is no need to consider these variations for the purpose of this research as the focus is on the shape of the graph and not on the meaning behind them. A list of target graphs is created to represent each possible known strategy. Each of these graphs will be unique, though they may share subgraphs. The target graphs are not isomorphic with respect to one another. There will also be a candidate graph. This candidate graph will start with only a root node at the beginning of any episode.

For example, in Figure 5.39, there are a number of positions being represented. Recall that each position, p, is both a location, l, and a posture, ρ. This figure shows each position with a vector indicating the posture (orientation, in this case) of each agent. These postures can then be used to represent a single agent as it moves from position to position via transitions. Recall that such transitions from one position to another are movements, m. Figure 5.40 shows these transitions, while Figure 5.41 shows these position transitions being translated into movements. While each node in the previous figure represented a position, and each edge a transition, this new figure encapsulates these into states as nodes and the transitions between as movements (the edges). It is now possible to represent an entire chain of movements (referred to

as a vector of movements) as a new graph, as described above, and shown in Figure 5.42. Now that the agent movements are captured in this vector of movements, shown as a graph, they can then be further encapsulated into a behavior. Figure 5.43 shows this encapsulation, so that now each node in the new graph can represent an entire sequence of movements as a behavior. Recall that a policy π is a mapping from an agent, o, to a behavior, b. This is shown as an encapsulation in Figure 5.44. Further, when policies are used to map the behaviors of an entire team of agents, as shown in Figure 5.45, we have finalized this hierarchical encapsulation into a strategy, σ. During the simulation, as the team progresses through the various stages, the ISSE will transition the strategy of the team. This creates a chain of policy progression from the encapsulation of the strategy mappings as shown in Figure 5.46 into the output of the ISSE, as shown in Figure 5.47. This procedural encapsulation leverages the hierarchical structure of strategy to simplify the resultant graph of every action for every agent on every team into a manageable (and thus, more easily recognizable) graph. This process can then be exploited, as observations are made throughout the simulation, to infer the strategy that another team is using based on graph matching, detailed next.

In a further example, we wish to consider a simulation of 5-vs-5 Speedball Paintball. This fast-paced version of paintball pits two teams against each other with a vast array of objects between them (half of a field is shown in Figure 5.48, the other half is a mirror of this one). All players must start in their home base and then progress from obstacle to obstacle while both avoiding enemy fire and attempting to

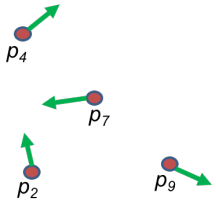


Figure 5.39: Observed Positions (locations with orientation)

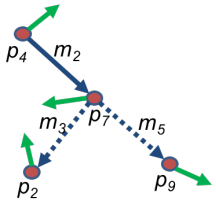


Figure 5.40: Positions with Transitions

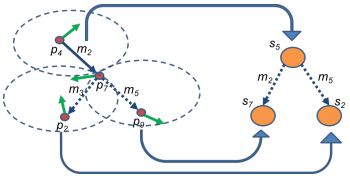


Figure 5.41: Positions Encapsulated as Movements

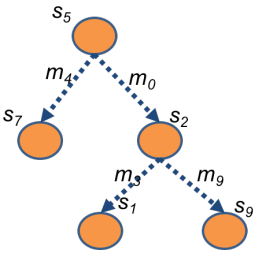


Figure 5.42: Encapsulated Graph of Movements

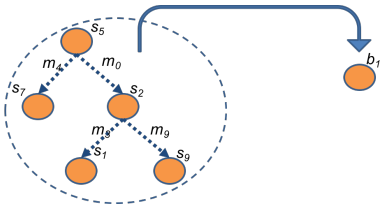


Figure 5.43: Movements Encapsulated as Behaviors

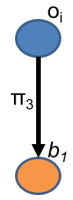


Figure 5.44: Policy Mapping an Agent to a Behavior

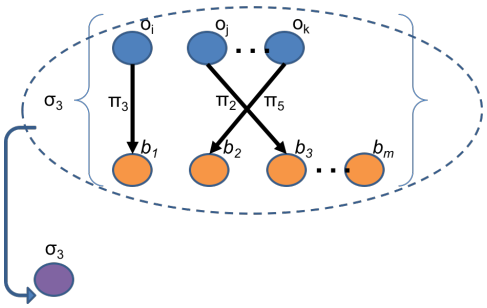


Figure 5.45: Strategy Mapping Policies to Team

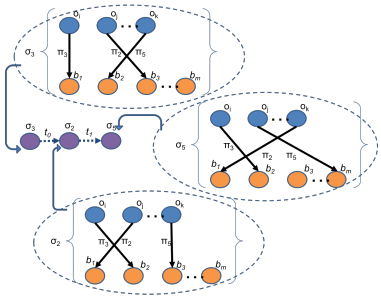


Figure 5.46: Strategy Progression for a Team

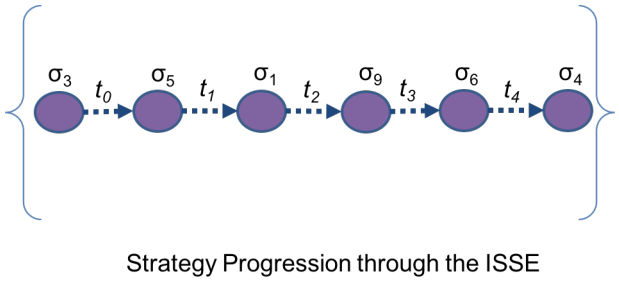


Figure 5.47: Final Strategy Progression

eliminate the opponents. While there are flags for each team to capture and return to base, thus ending the match, the majority of the matches end via team elimination. Recall that there are several distinct phases of each round of the game, but they can be reduced to offense, defense, and end game. Most of the teams will start out on offense and then shift to defense based on progress and eliminations. The decision to switch to end game is necessitated by the elimination of the majority of teammates. For the graph-matching algorithm, we decompose the multi-agent, multi-team strategies into a graph composed of the union of the various policies available to the particular strategy. This procedure, demonstrated in Figure 5.49, shows the progression from the field layout (with the position mappings) to the resultant graph. Each of the subfigures shows a policy (drawn from a particular strategy) being added to the diagram. This shows how the strategy graph is built. This complex graph is a directed graph but not implicitly acyclic. Each edge is weighted based on the probability of selecting this edge as a path for the agent to travel. This leads to the finite state automaton model representation of the strategy model. In this FSA, each policy is amalgamated into a multi-branch pathway that is a partial representation of

the strategy (and, as such, could be derived from any number of cases where there is a probabilistic progression of players through a space, mappings for data, a constellation of sensor inputs, or any other scenario that can be realized as a FSA). Recall that each strategy has several variables that define it along with a set of policies to which it has access. This set of policies is a subset of all available policies available to all strategies being followed by all teams in the simulation.

The simulation of the multi-agent, multi-team 5-vs-5 Speedball Paintball is written about in the Experiments section, Chapter 7, but some of the results are shown to prove that this graph matching approximation algorithm is both valid and useful in larger, more complicated environments. The results of these experiments will be reported separately in the Results section, but the approximation algorithm performed exactly as desired.

Graph Matching

During each episode the agent watching the other players will observe the states and actions of the players. To model this, the agent builds a FSA model as a graph that tracks not only states and actions, but the likelihood of any and all transitions (movements from one location to another, translated as traveling from one vertex to another along an edge). Each time the player makes a move (i.e., takes an action from the observed current state) the candidate graph is either expanded or updated. If the action has been observed before from this current state then the existing node is updated to reflect this event. This will help to build the probability model for

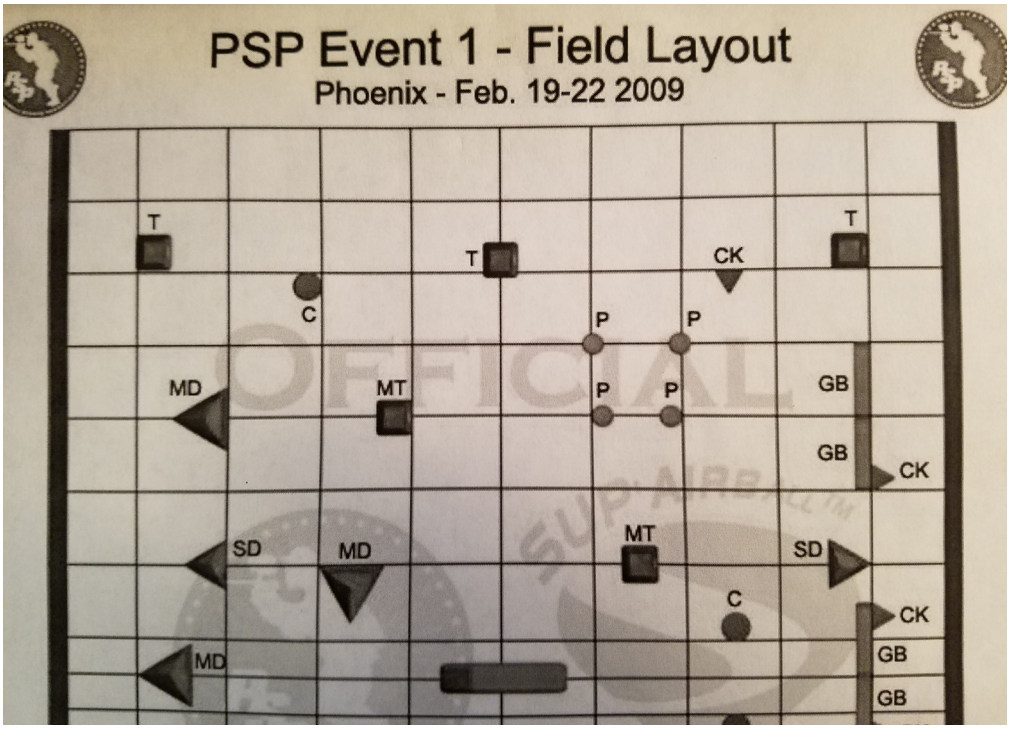
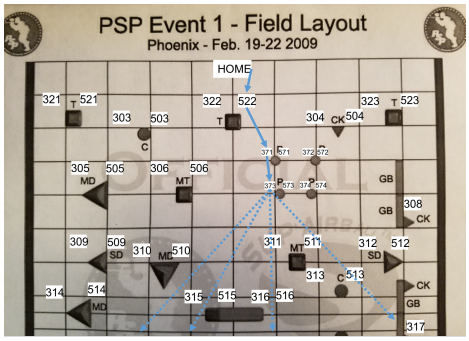
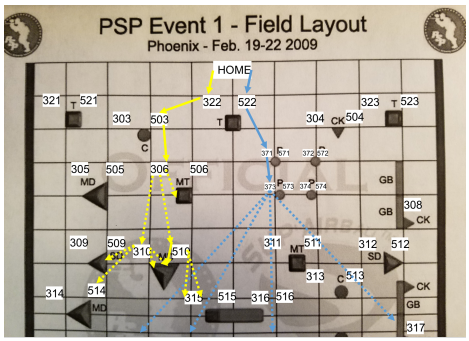


Figure 5.48: Official PSP Field

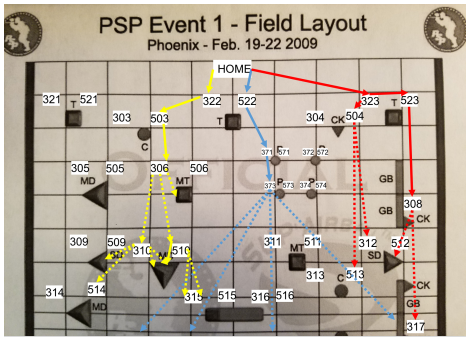
transitions from this state. If the state-action pair is new then a new node is created. The following series of diagrams (Figure 5.50) show this progression. The left-hand side of the bar shows the three sample targets (the graphs to which we are hoping to find a match). The right-hand side of the bar shows the candidate graph. As this candidate graph is being built, it is constantly being matched to the existing target graphs. There is a heuristic that scores the match based on various factors. This heuristic is adaptable to each situation, but it remains constant for any given implementation. The heuristic provides the overall quality of the matching of the two graphs — the currently considered target graph and the candidate graph. This process is repeated until each target graph has an approximation of how well it matches the candidate graph. This is the likelihood that this target graph is a match for the



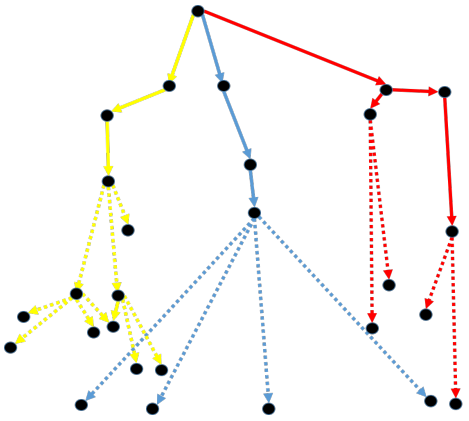
(a) Policy π_0



(b) Policy π_0,π_1



(c) Policy π_0,π_1,π_2



(d) Strategy σ_0

Figure 5.49: Building a Strategy Graph from Policies

candidate graph. These heuristic values become the seeds for the belief network. In general terms, the belief network is accumulating votes for each target based on the number and quality of matches it has with the candidate graph. As a result, the strategy inference algorithm is able to infer the most likely match, and thus the most likely strategy that the other players are following (the glow around the graphs in the figure shows their relative likelihood of matching, thus their belief that they are the best match for the candidate graph).

Of course, there is another possibility, namely that the strategy being observed (i.e., the graph being built through observation) is not in the library of known strategies (i.e., not one of the target graphs). In this case, the strategy inference algorithm adapts its behavior to perform two different objectives simultaneously: first, record this new graph and add it to the known strategies (i.e., it becomes a target graph in the next episode); second, begin to evaluate the scoring outcome of the player's candidate graph and the currently selected target graph (i.e., the one that the agent is currently following). This second objective attempts, in interactive time, to make sure that this new and previously unobserved graph is not better than its own strategy graph. If it is better, the agent can either swap to another strategy by evaluating the performance of all target graphs or begin to emulate the new graph it is observing from the player. *In either case, the computational complexity begins to stress the limits of the interactive time analysis requirement.*

An approximation algorithm can increase the performance of this strategy inference in both directions. First, it will have a higher success rate of interactive

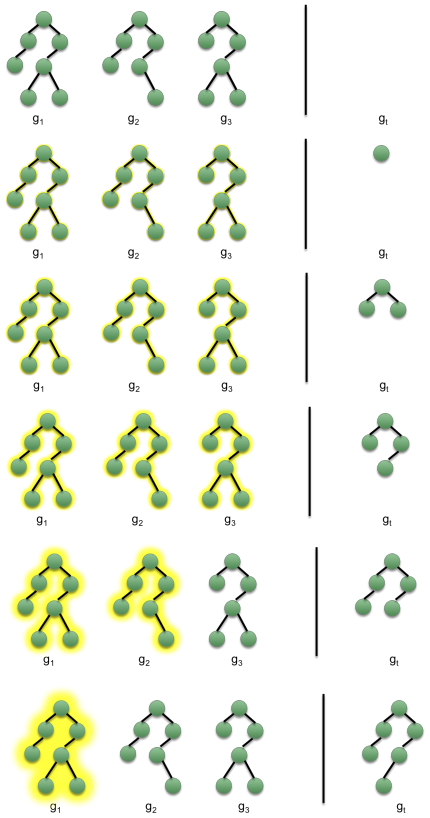


Figure 5.50: Approximate Matching

time graph matching as it considers more of the overall graph each step and not just the last observed state-action pair. Second, it recognizes the correct target graph for matching as it works to eliminate target graphs from consideration in the approximate pruning step of the algorithm. This process is shown in the series of figures in Figure 5.50. The approximation algorithm seeks an approximate solution through the dual process of pruning the target space and rapidly determining matches. To prune the search space, the approximation algorithm ignores some graphs based on the most likely eliminations (i.e., the most likely 'no' answers). As a result, the target pool is

reduced rapidly. Next, the algorithm searches the graph matching intelligently. Each method is described in more detail below.

The pruning process matches the most prominent features of each graph first, thus eliminating many graphs early in the process. In order, the algorithm asks several important questions: how many vertices? how many edges? highest degree vertex? next highest degree vertex? etc. Each of these questions eliminates both non-matching exact match target graphs and isomorphic variants of the candidate graphs as none of the questions deal with root node orientation or other metrics that would require identical graphs. This pruning is even more useful as the complexity of the graphs increases. As graph complexity increases, the likelihood that these quick determinants would match is even lower. As a result, the expectation is that the relative performance of the approximation algorithm will increase as the complexity increases.

The intelligent graph matching also seeks to rapidly determine 'no' answers. It must, in the worst case, perform the same edge-by-edge matching that the complete matching algorithm performs, but there are two factors that affect this. First, the order in which edges are considered is vital. The approximation algorithm orders these edges by the highest degree vertex. As a result, the approximation algorithm first considers the most difficult vertices and their corresponding edges. This is also subgraph isomorphic and isomorphic tolerant. Second, the algorithm exits on the first non-match and need not consider any additional vertices or edges after a non-match.

Many complete matching algorithms run through the entire graph either way. These two considerations work together to ensure optimal performance.

The Experiments section describes the experiments that were designed to test these hypotheses and gather performance metrics.

5.6.3 Implementation

To formulate the approximation algorithm the complete matching algorithm was transposed into an integer linear programming problem and, via a relaxation of the constraints, into a linear programming algorithm.

$$\text{Given: } Set(g_{targets}) \in G, g_{candidate}$$

$$\text{Find: } argmin_{g \in G}(\delta)$$

$$\text{Where: } argmin_{g \in G}(\delta)$$

$$\text{if } g_{target} = g_{candidate}, \delta = 0$$

$$\text{else } \delta \rightarrow min$$

This formulation asks for the target algorithms to be ranked by minimum δ so that the closest possible match to the candidate (i.e., the minimal δ differential) is the target selected. As noted previously, the approximation algorithm actually serves two simultaneous functions: prune non-matching candidates and accelerate approximate matching. These goals were realized in the experiments that follow.

Algorithms

There are three algorithms that have been implemented in this work. The first is the complete matching algorithm that matches every vertex, edge, and connection. The second algorithm is the initial approximation algorithm that only focuses on matching but with the elimination of invalid candidates (quick 'no' evaluation). The third algorithm is the improved approximation algorithm. This takes the initial approximation algorithm and improves it by quickly eliminating invalid candidates and performs in-line pruning to reduce the search space. Each of these are detailed in the chapter.

5.6.4 Experimental Results

This research proposes a system that matches strategy graphs realized as FSAs. In order to throughly test the robustness of the algorithm, it was first tested against procedurally generated randomized graphs. This creation process produces a large population of diverse graphs that follow stochastic branching, much larger and more complex graphs than may be found in standard scenarios. This process varies the depth, branching, and diversity of the graphs (according to the variables described below). This sample population had no guarantees of containing any matchings. There are several variables to the graph generation algorithm: *numgraphs*, how many graphs to generate; *gen*, how many generations in each graph (depth of the graph); *offspring*, the number of children to have at each level; *family*, the size of the set from

which the vertex labels are drawn. Each of these variables was randomized below this limit at graph generation. Once the graphs were generated, the algorithms analyzed the set and calculated the subgraph isomorphic matches, the isomorphic matches, and the total time (in seconds) to execute each algorithm. The results were then recorded and analyzed. This procedure will work with any connected graph structure, not just simple graphs, and so is widely applicable. This experiment proves that this matching of strategy graphs can be done in interactive time with a large population of complex graphs, even these highly-complex and purposely diverse graphs.

Alg	**Category**	**Trial1**	**Trial2**	**Trial3**	**Avg**
Heur	SI-morphs	278	289	327	**298.00**
Heur	I-morphs	78	87	92	**85.66**
Heur	Time (sec)	1777	1674	1689	**1713.33**
Aprx	SI-morphs	278	289	327	**298.00**
Aprx	I-morphs	78	87	92	**85.66**
Aprx	Time (sec)	173	168	169	**170**

Table 5.3: Heuristic and approximate graph Matching (n = 1000)

Graph Complexity	n	n^2	n^3
Sub-Graph Isomorphs	21	278	380
Isomorphs	1	78	10
Heur Match (time in sec)	427	1777	24310
Approx Match (time in sec)	9	173	111

Table 5.4: Growth of Algorithmic Methods by Complexity

In the second experiment, the algorithm was put to the test in the Roshambo example mentioned above. While this game is simple (that is the reason it was used), it is sufficiently expressive to test the solution. The results show the time to recognize the strategy is greatly reduced by using the approximation algorithm. Additionally,

the number of target strategies can be reduced because of the number of strategies that are isomorphic or subgraph isomorphic to another strategy. The results, shown in Table 2 of the final large-scale experiment (Chapter 7), reduce the number of recognition steps required to positively identify the strategy in play from hundreds of steps to less than 20 (about 13 steps, on average). The Roshambo results show the algorithm working in a real environment, but not a complex one. For the complexity proof we implemented this in a more difficult real-world example, following.

The third experiment increased both the complexity and the difficulty of the matching. To increase the complexity, we implemented this graph matching approximation algorithm in a multi-agent, multi-team simulation. In this simulation, 5 vs 5 Speedball Paintball (details are found in the final experiment documented in Chapter 7), there are 5 agents on each team, each following their own policy drawn from their team strategy. They are attempting to infer the strategy of the opposing team in real time so they can switch policies or strategies to better their position. The results show that such inference can be done within the interactive time limit constraint, shown in Table 5.5, with a significant increase in strategy recognition.

It may seem counter-intuitive that the heuristic matching, where every single vertex and edge are matched, would produce lower accuracy than the improved approximation method, but this is another side-effect of the complexity of the system. The heuristic matching takes so long to calculate that the 'world' has changed (i.e., this is a dynamic environment). As a result, the graphs that are being compared are becoming antiquated during the matching procedure. This was the inspiration for

Method	Correct	Total	Accuracy	Time(sec)
Heuristic	60	81	74.07%	1778
Heuristic	63	81	77.78%	2035
Heuristic	66	81	81.48%	1694
Approx	38	81	46.91%	5.3
Approx	35	81	43.31%	5.1
Approx	40	81	49.38%	5.8
Improved	**72**	**81**	**88.89%**	**5.9**
Improved	**77**	**81**	**95.06%**	**5.2**
Improved	**74**	**81**	**91.35%**	**5.7**

Table 5.5: Strategy Recognition in 5v5 Speedball

the goal of real-time graph matching, or at least interactive time graph matching. Further, not only did we achieve the matching, with even higher accuracy, but it was with a negligible time penalty. As you can see from Table 5.5, the simulation runs have an average run-time of 5.4 seconds, but adding in the graph matching only raises the average to 5.6. This sub-second result qualifies as interactive time, especially since the variance is so high in run-times. It should be noted that the simulation was run in serial to slow it down for analysis; in parallel the difference in timing is not easily measurable.

There is another aspect of this third experiment that amplifies the results from the second experiment (the Roshambo experiment). In interactive time considerations it is imperative that the algorithm utilize its belief network to constantly provide the most likely match evaluated so far. This means that another key element in evaluating the effectiveness of the algorithm is in how fast it can recognize the correct strategy. As it was with the Roshambo experiment, the Speedball Paintball experiment showed that the inference engine could recognize the correct strategy in an average of about

25 moves. A move is either a literal movement, a shot fired, or a player taking cover. Considering the complicating factor that all players start in home base (by rule of the game and the simulation), this shows that the recognition occurs very early in the encounter. Further, the fact that some players could be eliminated in fewer than 25 moves also makes the recognition (the graph matching) even more difficult. Table 5.6 shows the results of the experiments showing how many steps it took to recognize the correct strategy. Each time the inference engine selects another strategy as the most likely this counter is reset to the current step number. This means that even if the correct strategy is recognized early on but is discounted later on, even if the engine comes back to select this same strategy, the last time it is selected is the number of steps recorded.

Trial Number	**Recognition Steps**
1	22
2	27
3	24
4	26
5	25

Table 5.6: Moves to Recognize Correct Strategy

5.6.5 Conclusions

The experiments showed that the growth rate for the complete matching algorithm is exponential while the growth rate for the approximate solution is linear. The complexity of graph growth has a significant effect on runtime. Heuristic matching is, as the related works showed and the experiments confirmed, a taxing process on

a computing platform; with homeomorphic, subgraph isomorphic, and isomorphic variants the compute time exceeds reasonable computation time even in highly parallel environments. In multi-agent interactions, where interactive time decision making is critical, we have shown that the use of FSA-based Belief Networks, where this kind of approximate graph matching is required, is tractable. The algorithm also returns the most likely match observed thus far, allowing interactive time decisions in multi-agent scenarios without having to wait for episodic completion or complete matching. The experiments demonstrate that this FSA-matching, even approximately, is both interactive time and accurate enough for decision making even in multi-agent, multi-team environments.

Another surprising but welcome conclusion is that the pruning portion of the approximation algorithm is so effective that it keeps the growth rate constant within the linear boundary. This result is manifesting itself because as the graphs increase in complexity the unique features expand; thus, there are more graphs eliminated in the culling step. This means that the size of the target set is staying consistent and manageable. Again, the greater complexity works in the improved approximation algorithms favor as there are more 'no' flags that are triggered because of the higher probability of unique vertices, edges, and degrees.

The complexity analysis is shown in Table 5.7, where g is the set of graphs in the set to inspect and m the number of eliminations in the set of g:

This research has pushed the boundaries of graph matching to make it tractable in interactive time via an approximation algorithm. The results also show that there

Algorithm	Complexity
Graph Matching (pair)	$O(V^2E)$
Graph Matching (weights)	$O(V^2EW)$
Graph Matching (set of g)	$O(g^2V^2EW)$
Isomorphic Graph Matching	$2^{O(\sqrt{n}log^2n)}$
Approximate Matching	$O((g-m)^2V^2EW)$
Improved Approximate	$O((g-m)V^2EW)$

Table 5.7: Complexity of Algorithms

is no loss of precision with the approximation, though this was not a claim of the initial hypothesis. This additional benefit means that the algorithm achieves greater speed and lower complexity without compromising the integrity of the system. These results, in total, conclude that the improved approximation algorithm is a success and the hypothesis is confirmed.

Chapter 6

Overwatch: Strategy-based Multi-Robot Interaction Testbed

6.1 Overview

The Overwatch system creates a multi-faceted testbed for experimentation in multi-agent systems, with special emphasis on representing, implementing, and recognizing strategies. This is accomplished in a modular and multi-tiered approach that offers flexibility and is agnostic with respect to hardware, software, or other systems considerations.

In order to create an environment for interaction where this kind of strategic interaction can be tested it was necessary to design Overwatch. Overwatch is a real-time multi-agent system that allows simple, limited robots to behave intelligently and interactively. This is accomplished by creating an arena that is defined by the field of view of an overhead camera. This camera is the data source for the location, tracking, and directional information provided independently to each robot within the system.

The name Overwatch comes from the modern military and means a highly-trained unit that takes the high ground and provides cover, direction, and organization to an operation. To accomplish this, the overwatch identifies agents within the theater of

Figure 6.1: US Soldiers Takes Overwatch Position (Associated Press, 2012)

operation (both friendly and non-friendly) and then works to coordinate the individual actions of the smaller units as it observes their progress, the enemy locations, and the ever-changing flow of parameters that may amend their plans. This requires an understanding of the underlying training of the teams and how each team can and should be used. This relates to the state-action sets previously discussed and the policies which govern them. In this situation, the overwatch is exhibiting the strategy which governs the instructions given to each unit (these can be visualized as policies rather than actions as such action by action communications would overwhelm the bandwidth available for communications). This is a real-world argument for strategy. It can be observed that in theater operations currently underway around the world are using an overwatch strategy that confirms this exact experimental scenario such as the E-2 Hawkeye surveillance airplane that orbits on-station over conflict zones or UAV's that do the same in a more local context. These overhead 'cameras' provide

strategic oversight of the state of the zone that they are surveying and give real-time updates on friend and foe locations. This high-level coordination is the behavior being mimicked in the experiments documented here. In addition, the overwatch needs to be inferring the strategy of the enemy within their theater so that they can adjust their strategy as needed to match their observations. This maps well into the model considered in this research.

Similarly, Overwatch takes the high-ground as an overhead camera that can observe the entire operation space and can identify and track the individual agents within this space. Additionally, Overwatch can communicate independently with each agent. The system can, with each agent's identification, understand the multi-agent teams as well. This system provides sensor-limited robots (i.e., those without internal navigation capabilities or localization ability) the functionality of sensor-capable robots within this defined area. The constraint on the system is based on the field of view and the resolution of the camera, the size of the agents, and the size or capabilities of the marker system. If the camera has a wide field of view and high-resolution the size of the arena is larger. If the agents are smaller, more of them can fit within the arena. If the marker system is recognizable at smaller sizes (i.e., smaller than the footprint of the agent) then the physical size of the agent is the only size factor. If the markers are larger than the footprint of the agent, as observed overhead, it is the limiting size factor. The details of this system are covered under Methodology.

It is important that a distinction is made that the main thrust of this research, though realized in the Overwatch system, is not confined only to such a system. This work will present ideas that are equally applicable to sensor-capable or even sensor-rich robotic systems and will work across any scale wherein these same concepts (i.e., an arena, an overwatch capability, an individualized communication system, and centralized coordination) are available. This research just uses this scale for the purposes of cost and repeatability, but these concepts, once proven in this complex small scale, are not in any way limited to such a scale but are equally useful in a larger scale. This point will be reinforced as the research is concluded.

Additionally, it is noteworthy that Overwatch, because it has total knowledge of all agents, communications, strategies, etc., can then be constrained to simulate many different environments. One example would be to isolate the team communications from each other so that they are not aware of the internal processing of the other team's actions. Another example would be simulating a Master Controller where all information, communication, and strategy is centralized. Alternately, the opposite could also be enforced, that is that no agents have any internal information about any other agents. In short, many different paradigms can be configured and enforced within the Overwatch system. This leads to many possibilities for future research opportunities.

6.1.1 Robots

The agents in this scenario are Scribbler robots (Parallax, 2012). The Scribbler, from Parallax, Inc., is a sensor-limited robot having a few IR sensors, limited resolution wheel encoders, and some line-following IR sensors mounted on the bottom. This configuration is greatly enhanced with the addition of the IPRE board (IPRE, 2007) that mounts directly on the serial port of the Scribbler. This board provides light sensors, a color camera, and a Bluetooth-over-serial connection to wirelessly connect to the Scribblers. There are several LED's on both the robot and the IPRE board as well. These could conceivably be used for communication, but they are view-limited by their angle and the physical configuration of the Scribbler, especially from an overhead view. The Scribblers also have a very limited ability to store programming information so it is not practical for them to try to run their own independent code or manage their own inter-agent communications. In this research the central computer calculates all the information that the Scribbler will need and then sends only those control signals to each one. This shifts the sensor-limited robot into simulating a sensor-capable robot.

While there are several sensors on the robot, it does not have the ability to localize. It cannot return its x, y, z coordinates in real-space nor its θ heading. Because of this, it must rely on dead-reckoning or limited beacon guidance (e.g., the light from a flashlight or blob tracking) to navigate. Even this limited form of navigation is not true navigation - the robot is unaware of its current location, its location within a

world, or its heading. Instead, this just represents a type of reactive navigation like obstacle avoidance rather than true navigation where the robot is tracking along a map. Additionally, the Scribblers have limited reliability to track straight lines (given the differential of the low-cost motors) which is only exacerbated as the battery runs down (see Failure Modes). Also, the motors are attached to large, thin plastic wheels that are prone to slip, so their limited encoder data is also compromised. The robots have trouble initiating motion when the input is below a certain power threshold, so they require a surge to get started. All of these factors combine to show that the Scribbler, while a nice and inexpensive platform, cannot function as a reliable agent in a more complicated multi-agent system without some guidance. This guidance is provided by Overwatch. The Overwatch system provides the arena and multi-agent system that this research is based on.

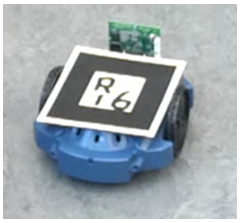


Figure 6.2: Scribbler with Fluke and AR Marker

6.1.2 Camera

Overwatch is enabled by having the high-ground view of the arena, so an overhead camera is used. The camera determines the size of the arena (its viewing angle or field of view). As the camera is raised farther from the floor, the same field of view

now encompasses a larger area, but the effective resolution of objects on or near the ground is diminished. The system is a balancing act across these two considerations - placing the camera as high overhead as possible while maintaining sufficient ground-level resolution. In this research a single overhead camera is used, but the system could utilize multiple independent cameras and merge their perceived worlds into one larger world to allow for an increased arena size while maintaining the same resolution. There are two different cameras used for these experiments, the Mobotix Q24 (Figure 6.3) security camera and a HD (720p) webcam (Figure 6.4). The security camera is a fixed position high-resolution mounted system that takes the image from a spherical lens and translates it into flat-view images and transmits them via PoE(power over Ethernet) as an HTTP stream. The webcam is not fixed and can be relocated to an overheard position, and angled position, or even a point of view(POV) position without compromising the performance of Overwatch.

Figure 6.3: Mobotix Q24

Figure 6.4: MS HD Webcam

In choosing a camera the effective resolution of the optics must be considered. The camera cannot be so far away from the arena floor that it can no longer provide images with sufficient detail to recognize the markers on the agents. Any camera can

be used in the system, but the system will be constrained by the quality and field of view of the camera.

6.1.3 Augmented Reality

There are many methods for using augmented reality(AR). The most common, by far, is ARToolkit. This toolkit is built upon the required OpenCV platform (which supplies the computer vision framework for interacting with the graphics hardware). Together, these two pieces of software allow the computer to interpret video, check for AR markers, and draw graphics on those markers. The original ARToolkit is no longer being developed (by the original creators) and has been supplanted by ARToolkit Plus. This newer version is a significant improvement over the original but it is less compatible across platforms. As a result, Overwatch is designed around ARToolkit so that it is maximally compatible. Additionally, if Overwatch works within the confines of the older system, it is provable that it will only work better on improved systems. There are other AR packages available which were also considered. The first was PyARTK, which works in Python. This toolkit was intended to be a port of ARToolkit that would work natively in Python. As it was problematic and unreliable in testing, and offered no additional functionality above ARToolkit, it was passed over. The next was OpenCV AR. This toolkit was supposed to be a native set of AR tools inside of OpenCV, but it was also functionally limited and relatively new. Due to its lack of robustness and functionality, it was passed over as well. There are also a few other lite versions of AR, like Aruco and Augment, that are designed to run on mobile

platforms (e.g., Android, iOS). While these may add functionality or expandability to Overwatch, they do not have the processing power and communications ability to control multiple robots and coordinate the communications necessary for the system.

6.1.4 Computing Environment

The computing environment also presents challenges to the system. The robots have a software platform, called Myro, that simplifies communications and commands with the Scribblers. The Myro software is written in Python, though it has been ported to C++ (J. R. Hoare and R. Edwards and B. MacLennan and L. Parker, 2011). The ARToolkit software is written natively in C, while the interaction with the arrays of robots and arrays of marker patterns is better facilitated in C++ (especially when considering these elements as objects in a multi-threaded environment). The computing environment must also be robust enough to support programming with multiple threads, large memory transactions, high-throughput algorithmic calculations, and high-speed graphics capabilities. Another consideration is the operating system. Windows greatly simplifies the coordination of multiple Bluetooth-enabled Scribblers, while Linux (and the MacOS) provide solid, if convoluted, packet swapping over these connections. With all of these considerations in mind, the goal was to have Overwatch be available without constraint, so it will work in Python, C, or C++, in Windows, Linux, or on the Mac. The code requires slight alteration, but is portable across platforms. This was in important goal of the system, to be platform independent.

6.1.5 Overwatch System

To facilitate these experiments the Overwatch system was created. The Overwatch arena created for this research is based on two different types of overhead cameras. The first is a security camera that links back to the central computer over Ethernet, so the camera images are streamed over http. The second is a high-definition webcam. This unit connects directly to the machines controlling Overwatch. The cameras are mounted to the ceiling at a height of 9 feet in the lab. This gives a field of view of about 12 feet by 10 feet and thus defines our arena. This camera could be raised further, resulting in a larger arena, but then the effective resolution of the optics must be considered. The camera cannot be so far away from the arena floor that it can no longer provide images with sufficient detail to recognize the markers on the agents. As introduced previously, this limitation on the system can be traded off with higher-resolution cameras or larger markers. For these experiments the markers were produced from the ARToolkit. This augmented reality toolkit, produced by the University of Washington, is public domain and available for free. There are many such tools available, and each will be discussed in the methodology section. The augmented reality tools provide marker recognition and can be used to determine the marker's (and thus, the agent's) position in a relative camera space. There is much detail to how this orientation and subsequent navigation actually works, which will be given in the methodology, but suffice it to say for now that this is not a trivial process. The agents in this scenario are Scribbler robots. The Scribbler,

from Parallax, is a sensor-limited robot having a few IR sensors, limited resolution wheel encoders, and some line-following IR sensors mounted on the bottom. This configuration is greatly enhanced with the addition of the IPRE board that mounts directly on the serial port of the Scribbler. This board provides light sensors, color camera, and a Bluetooth-over-serial connection to wirelessly connect to the Scribblers. There are several LED's on both the robot and the IPRE board as well. These could conceivably be used for communication, but they are view-limited by their angle and the physical configuration of the Scribbler, especially from an overhead view. The Scribblers also have a very limited ability to store programming information so it is not practical for them to try to run their own independent code or manage their own inter-agent communications. In this research the central computer calculates all the information that the Scribbler will need and then sends only those control signals to each one. This shifts the sensor-limited robot into simulating a sensor-capable robot. The Overwatch system provides localization, vision, direction, and coordination capabilities. The power of Overwatch is seen here - a swarm of relatively inexpensive and small robots can be used in a powerful and coordinated away with little to no additional expense. This resultant system is a powerful tool to analyze swarms, multi-agent systems, cooperation vs. competition, and more. The Overwatch system provides the arena and multi-agent system that this research is based on.

Overwatch creates an arena that is defined by the field of view of an overhead camera. The camera is mounted to the ceiling at a height of 9ft in the lab. This gives a field of view of about 15ft by 12ft and thus defines our arena. For these experiments

the markers were produced from the ARToolkit (ARToolkit, 2008). The augmented reality tools provide marker recognition and can be used to determine the marker's (and thus, the agent's) position in a relative camera space. The system creates an array of robot objects that allow for the robots to be treated just as they would be treated if they were individuals. The addition of loops facilitates the single-robot programs in working with these multi-robot arrays.

In multi-agent systems multiple entities are introduced that are also working within the same environment, each with their own goals and motivations. It was common in past research for the focus agent, the one currently being considered, to view these other agents as part of the environment (Russell and Norvig, 2003). More recently work has been done in better representations of multi-agent systems, like (Russell and Norvig, 2003). These theoretical systems are not as frequently implemented as they are proposed, largely due to the cost of implementation and the availability of enough robots to test sufficiently. For this research, it is vital that these other agents be considered as 'intelligent' entities interacting with the environment in dynamic ways.

The Overwatch system works as follows: first, the system creates a data structure that holds the robot objects. Second, it creates a data structure that holds the various marker patterns that will identify the robots, targets, and other objects within the arena. Third, Overwatch uses ARToolkit to recognize markers by searching the current video frame for large black squares. Any such square will get the attention of ARToolkit. Once the target is acquired, it is inspected for the predetermined patterns

and is matched on the greatest matching likelihood (Figure 6.2 shows such a marker). The pattern is then 'recognized' and is entered into the visible marker array. Once all markers within the frame have been located, this array is returned to the code for processing. This processing is integral to Overwatch. Fourth, each marker is compared with its target marker to determine location and orientation. Fifth, the markers (i.e., the robots) are checked for collisions with targets or other markers and the traffic is directed according to the current strategy of the system. At this point, additional considerations can be made, such as organizing the agents to avoid collisions with teammates based on priorities, changing the target being sought, or implementing a new strategy based on the current information from the environment.

The process of checking each marker for location and orientation is complicated by the various coordinate systems. First, the markers are recognized in the camera coordinate system. Next, the robot has to be localized in the marker coordinate system. Finally, to achieve any reasonable navigation within the real world each of these markers has to be located in a shared real-world coordinate system. To accomplish this, the first marker, the robot, is considered. Its location is determined relative to the camera (Figure 6.5). In this coordinate system the orientation is figured as 0° for straight ahead, $+90^\circ$ to the left, -90° to the right, and $\pm 180^\circ$ for the rear, as noted around both the robot and the target (which is inverted).

The second marker, the target, is then similarly located relative to the camera. To determine the angle of orientation, θ_r, in ARToolkit, the marker is given an angle as if it were translated on top of the camera. This angle is skewed with the perspective

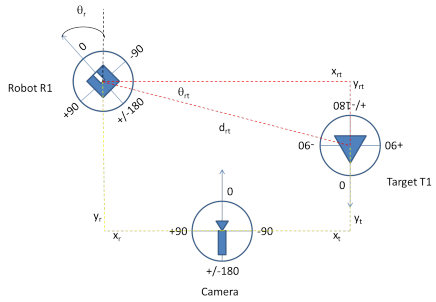


Figure 6.5: Camera-Space Coordinate System

of the camera, so this angle is biased. This angle needs to be corrected to give a real-world orientation so that a bearing angle to the target can be determined. To do this, the relative angle from the robot's marker to the target's marker, θ_{rt}, is calculated using trigonometry (Equations 6.1 and 6.2). This translates the camera coordinate system into a marker coordinate system (Figure 6.6).

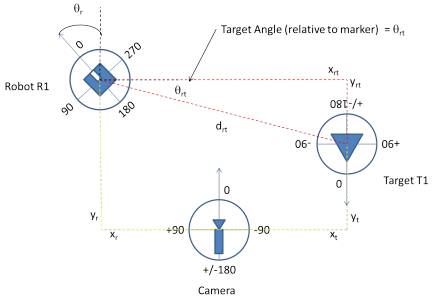


Figure 6.6: Marker-Space Coordinate System

With the camera-relative angle and the relative angle known, the bearing angle, $\theta_{bearing}$, can be calculated by the difference of these angles (Equations 6.3, 6.4, and 6.5). This gives a bearing angle for the robot to head towards the target figured optimally to turn in the shortest direction. This final transformation moves the marker coordinate space into the real-world coordinate space (Figure 6.7).

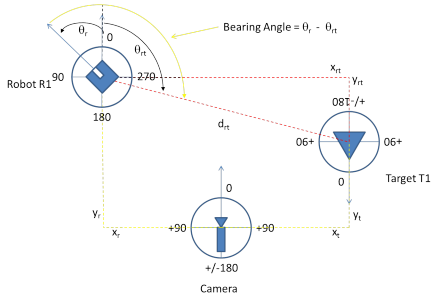


Figure 6.7: Real-Space Coordinate System

In this final space the angles are measured in a full circle, 360°, as opposed the the split ±180° system of the camera and marker coordinate system. This makes navigation identical for each marker, so they can now understand where they are and where any other markers are. If the robot is a tank-steer, it can simply rotate to this angle and proceed straight to the target. If it is conventional steering, a path to the target is calculated using speed and turn angle. In either scenario, or even with different robots with different steering paradigms, Overwatch can calculate this and compensate in real-time. Because these calculations are done in real-time and updated constantly, the system is robust to robot 'wandering' (where the motors are misaligned or aren't calibrated), moving targets, and collisions which move the robot off of its path.

$$\theta_{rt} = \tan^{-1}\left(\frac{y_r t}{x_r t}\right) \tag{6.1}$$

$$\theta_{r(corrected)} = 2\pi + \theta_r \tag{6.2}$$

$$\theta_{bearing} = \theta_{r(corrected)} - \theta_{rt} \tag{6.3}$$

$$\theta_{bearing} = \theta_{bearing} mod 2\pi \tag{6.4}$$

$$\theta_{bearing} = \begin{cases} if \theta < \pi, & \theta \\ else, & -(2\pi - \theta) \end{cases} \tag{6.5}$$

6.2 Experiments: Overwatch Implementation

There are three claims that needed to be tested. The first is that Overwatch is accurate and consistent in localization and navigation. The second is that it can control multiple robots simultaneously and efficiently. The third is that Overwatch can coordinate strategic behavior within a team of robots. One additional goal is to show scalability and robustness through each of these experiments.

6.2.1 Experiment 1: Overwatch vs. Blind-Reckoning

The first experiment is navigation and localization using Overwatch vs. blind-reckoning. In this experiment the blind-reckoning robots are driven, by odometry, forward for 6s, turned right for 0.45s, then forward for 3s, then another right turn for 0.45s, then forward for 6s. This should result in the three sides of a rectangle that ends back in line with the target, displaced by about 3ft. The target zone was determined

in a way that favors the blind-reckoning: three runs were made as directed, then the average finish point was used as the target location. Once the target was placed at this location, this experiment was run 7 times for each control method.

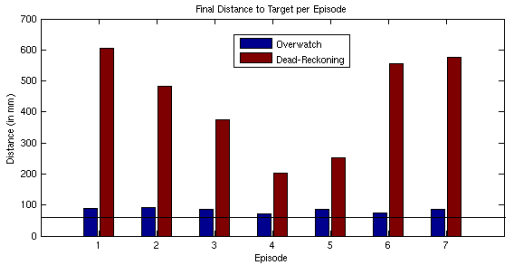


Figure 6.8: Final Distance: Blind-Reckoning vs. Overwatch

The results show, in Figure 6.8, that the blind-reckoning method averaged 435.86mm, with a σ of 149.23, from the center of the target. The Overwatch runs averaged only 83.14mm, with a σ of 7.06, from the center. It should be noted, as indicated by the line in the graph, that the center of the target is 64mm from the edge at its closest point, and 88mm from the corners, so the Overwatch navigation ended, as directed, at the edge of the target.

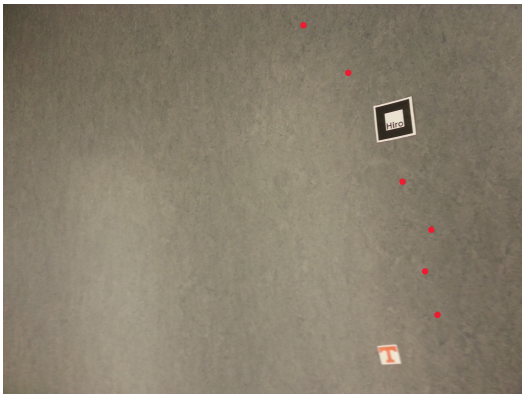


Figure 6.9: Accuracy with Blind-Reckoning

Figure 6.10: Accuracy with Overwatch

Figures 6.9 and 6.10 show the accuracy of the two methods by comparing their path endpoints. The blind-reckoning endpoints were spread over a much larger area and were farther from the center of the target. The Overwatch path results were grouped tightly at the target boundary. It should be stated again that Overwatch was only aiming for a proximity to the target, not a specific point or alignment, but it is still very accurate.

Making a comparison between the two control scenarios, blind-reckoning and Overwatch, is difficult because their path planning is different. These experiments show both the issue with unguided robots and the power of the solution system. It would be more informative to show a path error, but that is challenging with such different paths for the two methods. A visual must suffice. Examining the traces shown in Figures 6.11 and 6.12 shows the difference of the two methods. The traces all start at the starting beacon, in the lower right of the image, and are headed towards the target, in the upper right. In the Overwatch figure, 6.12, the first two legs were dead-reckoned to simulate the same conditions for comparison. Overwatch would never have wandered that far from the target to find its way back. The dark-colored trace is the common path, and the others are the traces while under Overwatch control. These traces show that the dead-reckoning pathways were inconsistent across each run, whereas the Overwatch pathways were consistent.

These experiments show at least two critical points: first, they demonstrate the overall inconsistency of blind-reckoning in sensor-limited robots (these runs were with fresh batteries; the same trials with older batteries were even less consistent); second,

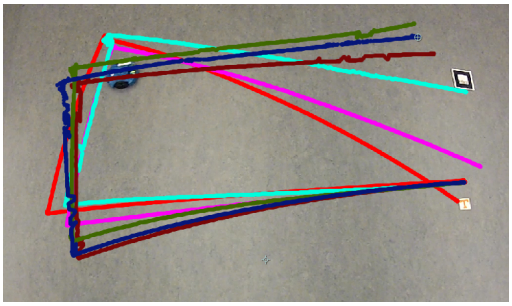


Figure 6.11: Blind-Reckoning Traces (7 Trials)

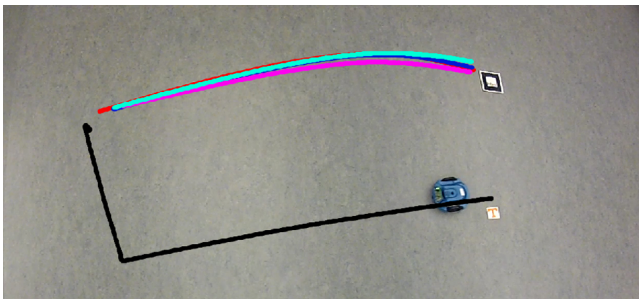


Figure 6.12: Overwatch Traces (7 Trials)

they show the difference that Overwatch can make not only with accuracy (distance to target) but with consistency (σ of distances to target) and proper path planning.

6.2.2 Experiment 2: Scalability

In the second experiment, multiple robots were tested on multiple teams for scalability and reliability. In these experiments, the system was running in C++ and Python, in Windows and in Linux. To test the scalability, multiple robots were run from one machine. For this experiment an arena was constructed that enclosed two teams of robots of 6 robots each for a total of 12 robots. These robots were each running an obstacle avoidance program.

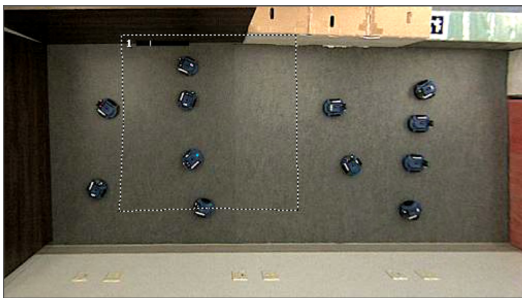


Figure 6.13: Two Teams of Six: Start

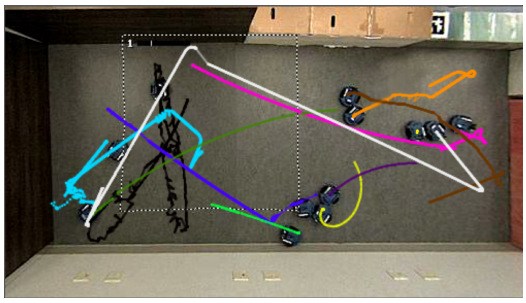


Figure 6.14: Two Teams of Six: End

The traces show that each of the robots was moving and being controlled by the system. In this scenario, the robots were passing back their IR sensor data to Overwatch and it was interpreting this sensor data and sending back steering instructions. These robots were not being tracked yet, just being controlled to test the throughput of the system and the control methods. Further testing of this scenario showed that the system works with up to 7 robots per machine, on 3 separate machines simultaneously for a total of 21 robots interacting in the arena. For reliability we can see that each robot stayed under system control, even when the robot itself was having issues (see Failure Modes).

6.2.3 Experiment 3: Strategic Team Coordination using Overwatch

The third experiment was designed to show that Overwatch can coordinate the behavior of a team intelligently. The scenario envisioned was one of a squad of 5 robots where 3 of them are seekers and the other 2 are 'on-station' as defenders. The 3 seekers are tasked with finding the target and converging there. When the seekers

all arrive at the target the entire team should disperse. Figures 6.15, 6.16 and 6.17 show the frames from the video of this experiment. The initial starting positions of the robots is shown in Figure 6.15. Figure 6.16 shows the traces of the robots from the starting positions to the point at which the seekers converge on the target. Finally, Figure 6.17, shows the traces as the robots disperse. These traces show that these simple robots can exhibit complex and coordinated behavior.

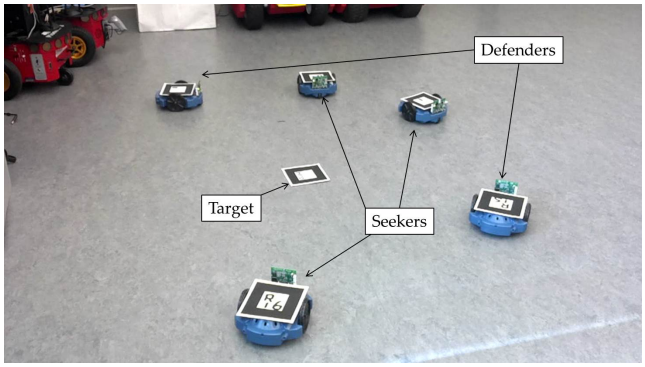


Figure 6.15: 5-Robot Trial: Initial Position

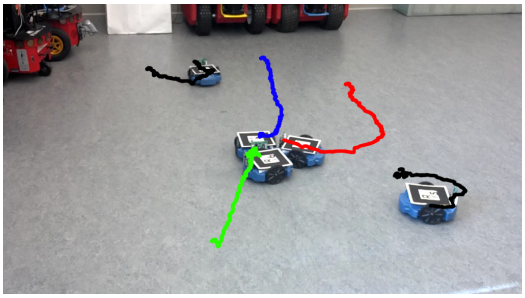


Figure 6.16: 5-Robot Trial: Seek and Defend

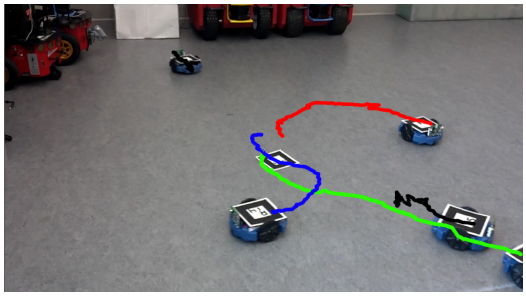


Figure 6.17: 5-Robot Trial: Disperse

6.2.4 Additional Experimental Outcome: Course Correction

One additional benefit that came out of the experiments was the proof of the Course Correction capability of the Overwatch system. The video of this experiment shows

several instances of the robots slipping, colliding, being pushed, and wandering. The real-time tracking of the targets and robots allowed for correcting motor wandering (the inability for simple, light robots to drive in a straight line, especially as the battery fades), external interference through collisions, and slippage due to a lack of friction with the surface. In each case, the disturbed orientation and location of the robot was, in Markov fashion, observed in its current state, and only this current state matters. The robot is simply given new directions from this orientation and locale to navigate to the target. Similarly, if the target moves, the course correction is automatic as the new frame is captured, the new location of the target determined, and the navigation instructions are issued.

Chapter 7

SiMAMT in Action: 5-vs-5 Professional Speedball Paintball

7.1 Introduction to Multi-team Multi-agent Experiment

This experiment seeks to consolidate the various elements of the SiMAMT Framework and apply them to a true multi-team multi-agent environment. The context for this experiment is Professional Speedball Paintball (PSP). These contests pit two five-player teams against each other on an enclosed field. Their objective is to engage the other team in an attempt to capture their flag and return it to base. To complicate this objective, the teams seek to eliminate the other team's members by tagging them with a paintball (a player is removed from the match when tagged by a paintball). As a result, these teams must think and act strategically and adjust their policies in real time. There are several types of players, roles, techniques, and tactics to consider as a part of this overall strategy ((AllExperts, 2016),(SpecialOpsPaintball, 2012)). These are described below.

The first players that are closest to the opposing team (at the front of their side of the field) are called the frontmen, or fronts. Their job is to advance as far up the field as possible as quickly as they can to establish a strong frontline. A team must make

many choices about how many frontmen they have, how quickly (or aggressively) they advance, and what their primary role is once established (like offense or defense). The next players are the mid-players. They are lined up behind the frontmen one or two obstacles back. The mid-players need to also advance, though neither as far nor as quickly, to establish strong mid-field positions in an attempt to block the progress of the other team's frontmen. They play a key role in preventing the other team from establishing their formation and enacting their strategy, as well as creating a strong defense of the heart of the field. Behind these players, closest to the home base, are the back players, or backs. The backs are vital in the initial break from the home base when each of the other players are attempting to establish positions. The backs must lay down suppressive and aggressive fire to disrupt the other team and cover their own team. These players play a vital early role and are often among the last players standing in the game. In addition to these positions there are also inserts. These are players who start in one position but then move forward to take on other positions as players on their team are eliminated. There are also some single agents within the team that take on specific roles, like advancing rapidly (and dangerously) up the side of the field. These will be addressed in the roles listed below.

There are also several situational roles that are vital to a team's success on the field. First, there should be a commander. This might be a coach, manager, alpha player (the highest scoring player of the match), or leader. The commander has the pivotal role of establishing and executing the team's strategy and overall gameplay. This is an ever-evolving action as the game progresses. Next, there are close combat

players. Their application is in the tighter areas of the course where the action is the most furious. These folks must move quickly and follow orders from the commander without hesitation. The complement to these close combat specialists are the riflemen. These players are often in a standing position with lower mobility and use longer range gear with higher accuracy at the expense of rate of fire. Their job is often on defense as they are watching the other team closely while covering their own players. Augmenting these rifleman are the heavy rifleman whose job it is to provide suppressive fire in an attempt to limit the mobility of the opposing team. The benefit is that it freezes the other team but at the cost of ammo. Finally there are snipers. Their role on the field varies from high-speed aggressive moves to flanking to stealth. In each of these maneuvers they are expected to fire less often but with greater precision. They do not want to give away their carefully obtained position unless they are confident that they can eliminate a player from the other team. These various roles are vital to the team's success and the strategy of the team must incorporate these roles and associated tactics while adjusting the level of aggression and the overall demeanor of the state of the game.

There are a variety of policies (tactics in paintball parlance) and strategies in paintball. Some example policies and strategies, from (Sports, 2013), are listed below:

'The Farrier' is a maneuver designed to draw opposing players into a U-shaped ambush where they are under fire from three sides and often cannot escape without losing 75% of their numbers, if not more. In the precursor to many commanders' strategies, the teams playing the game meet in battle lines, usually one battle line

for two teams, two battle lines for three teams, et cetera. Once the line of battle has been established and the team has engaged the enemy, a section of one team's line may withdraw after a particularly heavy bout of fire, feigning a regroup. The forces on the opposing battle line surge into the gap, thinking they are pursuing a weakened enemy who should be further attacked until annihilated. However, they have simply walked into a trap, and come under withering fire from three sides. Normally such firefights end very rapidly, often lasting only twenty seconds if not less. At the end of the firefight, either the remnants of the enemy have retreated, or they have been annihilated completely.

The 'roll-up' is the tactic of either maneuvering past the enemy's flank and attacking it from behind in coordination with friendly forces immediately opposite them, and then speedily proceeding down the enemy's line to attack and destroy as much of the enemy team as possible before a counterattack is executed. One common counterattack is to sap forces from across the line and form a second, shorter line, perpendicular to the first, which will meet the enemy flankers in a more even situation. To avoid the possibility of this counterattack, commanders may send a portion of their flankers well ahead of the others. These advance flankers continue on stealthily, usually a good distance away from the enemy line, so that when such a counterattack is executed, the advance flankers can swoop down and hit that new line's flank or rear, decimating their numbers and rendering the enemy even more vulnerable.

A 'solid attack' is the concept of attacking with essentially the commander's entire team in a slow, methodical advance toward enemy positions (hence 'solid', which

implies density and slowness of movement). Solid attacks are best employed if the commander has ambush elements out to either flank, such as marksmen, due to the vulnerability of solid attacks to enemy flanking maneuvers. Solid attacks are useful in games where the objective is stationary and close at hand. This is because solid attacks force one's team to clump up. In all but the most disciplined and elite teams, the commander loses each squad's congruity as the squad members mingle during firefights. To re-form, the commander must call each squad back singly, via radio or shouted callsigns, and then re-deploy them accordingly. Solid attacks can be countered by fast and mobile strike elements that mill around the slow-moving formation, striking randomly and harassing the enemy. Once the formation is somewhat disorganized (sometimes a difficult feat, as most elite teams resemble a rock in this fashion), then a large attack element attacks the formation's weakest point and, ideally, breaks it. Due to the solid attack's cumbersome nature, it is often combined with other strategies, such as the staggered attack or multi-prong attack, to get into a position where the solid attack may be employed with benefit. The best fit for this job are small people that are stronger because they are harder to hit.

'Hit and run' strategy is very simple: it is the concept and application of hitting the enemy hard and fast in a largely unpredictable place and withdraw before substantial resistance or damage to one's own forces can be brought about. Hit and run is also a strategy of attrition, in that it wears down the enemy forces gradually. Hit and run strategy is generally useful as two things: a delaying action, or one used by a weaker force. As a delaying action it can be very useful, especially if the enemy is

hit in crucial points. While stealth is not a necessity for hit and run attacks, it is beneficial to use stealth, as an enemy who is not aware of one's presence is likely to be more surprised when he or she is hit. Consequently, his or her forces will not be marshaled properly and will likely sustain greater damage, and additionally, the enemy will take longer to come about to face the attackers with great strength. Hence, it is generally a good idea to maintain stealth before and directly after an attack. Attackers withdrawing from the enemy usually withdraw backwards to regroup in a clump before maneuvering to attack again. Most commanders expect this, as it is a fast maneuver, although usually poorly disciplined as paintballers rarely like having to retreat. This is bait most delicious to many commanders. To exploit this, a force retreating from an attack can assume the Farrier tactic, which is to exploit the enemy commander's opportunism and draw the pursuing forces into a U-ambush in which the numerically smaller force can annihilate them. After this, the ambush forces flee to avoid the enemy's retaliation. As mentioned above, hit and run is also a useful defensive strategy. During a woodsball game, circumstances often force one's attack to, out of necessity, lapse into a defensive force that falls back on friendly territory and/or positions. This is usually for support reasons, either in support for friendlies in one's own territory or positions, or due to an attack's need for further support (if an attack is beaten back with heavy losses, for instance). A defensive force withdrawing slowly has the great advantage of concealed ambushes - in essence, a planted butter advance. Ambushes consisting of well-camouflaged paintballers who excel in extracting from enemy territory, usually marksmen, are set up in the rear

of the withdrawing force, while the rest of the withdrawing force stalls the enemy. Once the ambushes are in place, the team withdraws over and through them, leading the enemy into them. After the ambushers are fifty or so feet behind enemy lines, the withdrawing force resumes the attack and the ambushers suddenly come up and attack the enemy from behind, eliminating as many of the enemy as possible in their drive back to the rest of their team. Upon linking up, the team can either continue the withdrawal or execute a different tactic or a different strategy altogether, while facing a much reduced enemy force.

A successful strategy, then, will implement these various strategies when the situation calls for them according to a number of variables (such as aggression, offense or defense, or desperation). There are times when the way that a player moves is determined by both the strategy in place and the current policy (Muhlestein, 2014).

7.2 Specification

As provided by the strategy-based system background information in Chapter 3, this experiment implements that system. Details are largely repeated here for clarity.

The agent, o, is represented by a tuple, $o(\phi, \psi)$, where ϕ represents the limitations placed on the agent by the simulation and ψ represents the performance variables of the agent (Equation 7.1). The set of all agents within a system is O, so $o \in O$.

$$o =< \phi, \psi >, o \in O \tag{7.1}$$

The field upon which the simulation takes place is divided into positions that an agent can occupy. Each position, p, is a tuple, $p(l, \rho)$, where l represents the location on the field (referenced to a mapping of the field) and ρ represents the posture of the agent at that location (Equation 7.2). These positions, p, are drawn from the set of all positions, P, such that $p \in P$.

$$p =< l, \rho >, p \in P \tag{7.2}$$

These positions comprise the base element that is the state, s (Equation 7.3).

$$s =< p_n, p_{name}, p_l >, s \in S \tag{7.3}$$

These states can then be encapsulated inside of movements. A movement, m, is defined by Equation 7.4, Illustrated in Figure 7.1.

$$m : S \rightarrow S, \text{subject to } p'_n, m_s, m_p \tag{7.4}$$

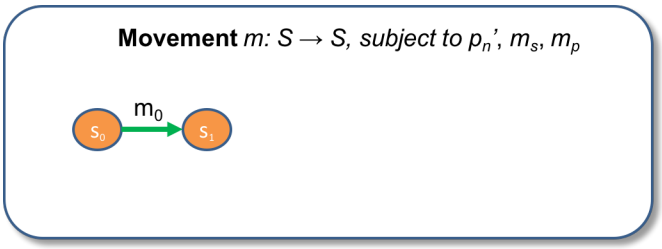


Figure 7.1: Movement Diagram

These movements can be joined together to form a behavior. The behavior is found in Equation 7.5 and illustrated in Figure 7.2.

$$b =< M, T, S, s_0, \delta, \omega, F >, b \in B \tag{7.5}$$

where

- M is the input set (set of all possible movements)
- T is the output set (set of all state transitions and observations from the agents)
- S is the set of states (set of all possible positions)
- s_0 is the initial position of the agent ($s_0 \in S$)
- δ is the transition function ($\delta : S \times (M \cup \{\varepsilon\}) \rightarrow \mathcal{P}(S)$)
- ω is the output function ($\omega : S \times (M \cup \{\varepsilon\}) \times S \rightarrow T^*$)
- F is the set of conditions of completion for the agent

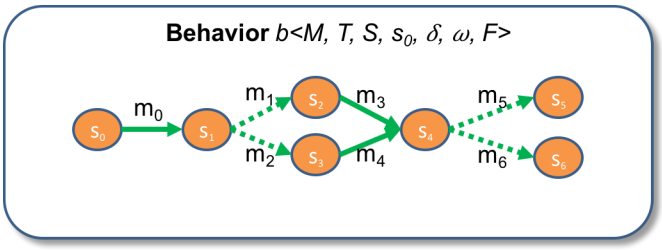


Figure 7.2: Behavior Diagram

These behaviors are assigned to particular agents via the policy, π (Equation 7.6, Illustrated in Figure 7.3).

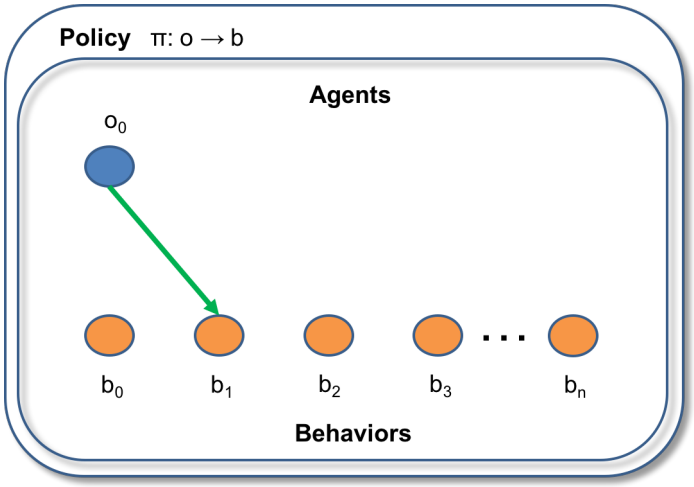


Figure 7.3: Policy Mapping

$$\pi : o \rightarrow b \tag{7.6}$$

A strategy, σ, is the mapping of all agents, O, within a team, τ (Equation 7.7), to their selected policies (i.e., the subset of policies that are available to the strategy (Equation 7.8)). The mapping is shown in Equation 7.9, and as illustrated in Figure 7.4.

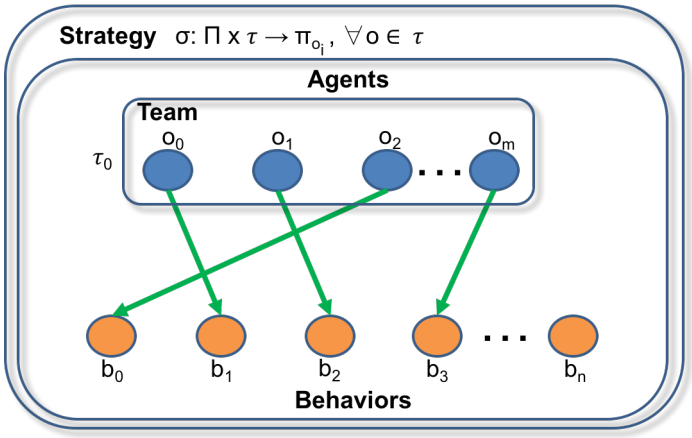


Figure 7.4: Strategy Mapping

$$\tau = \{o_0, o_1, \ldots o_n\} \tag{7.7}$$

$$\Pi_s = \{\pi_0, \pi_1, \ldots \pi_j\}, \Pi_s \in \Pi \tag{7.8}$$

$$\sigma : \tau \times \Pi_s \rightarrow \pi_{o_i}, \forall o \in \tau \tag{7.9}$$

The strategies are implemented by the Intelligent Strategy Selection Engine (ISSE), given by Equation 7.10 and illustrated in Figure 7.5.

$$ISSE =< A, \Gamma, \Sigma, \sigma_0, \delta, \omega, F > \tag{7.10}$$

where

- A is the input set (set of output from the SIE and the EE)
- Γ is the output set (set of all simulation outcomes)
- Σ is the set of all states (set of all strategies)
- σ_0 is the initial strategy
- δ is the transition function ($\delta : \Sigma \times (A \cup \{\varepsilon\}) \rightarrow \mathcal{P}(\Sigma)$)
- ω is the output function ($\omega : \Sigma \times (A \cup \{\varepsilon\}) \times \Sigma \rightarrow \Gamma^*$))
- F is the set of conditions of completion for the simulation

7.3 Implementation in the SiMAMT Framework

This specification is implemented in the SiMAMT framework by following the detailed breakdown offered earlier in Chapter 4. Each section is described in detail below.

7.3.1 Strategic Modeling

As previously mentioned, the FSA for each agent makes decisions at each interval. Based on this FSA, the various actions vie for the attention of the agent. Let's

Figure 7.5: Intelligent Strategy Selection Engine

consider the actions in turn. First, the player examines its policy to determine the next **move**. This is done probabilistically based on the policy (that lists each next move and its accompanying probability). Next, the probability of a move is based on the Movement feature of the strategy. If a move is issued, the player proceeds to the next position at the speed dictated by the policy. It should be noted that the speed with which they move determines their vulnerability during the move and the ability to aim during the transition. This provides an added realism to the simulation that is sensible (e.g., if you sprint, you are less likely to be hit, but less likely to aim correctly). The next action for consideration is **defense**. The probability is based on the Aggression and Posture factors. Again, under cover the agent is unlikely to be hit but unable to fire or make active observations. The next probable action to consider

is **offense**. This probability is based on the strategy factor of Aggression and Posture and metered by the player type and policy. There are a number of variables that affect this decision. First, the rate of fire is determined by the player and the player type. The fire probability is measured once per round fired. Second, the strategy factors of Distribution (for fire for effect), Aggression, and Posture weigh in. Additionally, the policy itself speaks to the firing positions and lanes where the player can fire. Finally, there are a number of global simulation variables that determine if shots fired actually hit or affect the target (e.g., general accuracy of paintball guns, probability of breakage of the paintball, etc.). These provide a stronger level of realism for the simulation. The next action to consider is **observation**. This involves the same basic flow as the firing phase, but they are only noting which players are visible and noting any transitions. If they are being fired upon then these positions are inferred to contain other players and are so noted (this uses an inverse kinematic to determine reverse trajectories based on those positions observable from this current position). This completes the action selection FSA for this player and the simulation moves on to the next. The Evaluation Engine (EE) evaluates policies and may suggest a policy switch to another policy within the strategy based on simulation results thus far.

To better understand the strategy factors involves examining them in context. For example, the Aggression speaks to how the strategy might overwhelm the policy to force a player to move where they might not have otherwise. It may also overwhelm their choice to take cover. Likewise, the Movement factor controls how likely they are to move and also in which direction. For example, the players may start to retreat

(move backwards through their move list) as the number of active players remaining on their team diminishes. The Distribution often overrides move probabilities to move players closer to each other or farther apart depending on this setting. The Posture factor controls fire aggression, movement aggression, and likelihood to stand and fire vs. retreat. Finally, there is a Persistence factor that keeps the players on their current policy or frees them to move to another policy. These factors, as well as the subset of policies, differentiate the strategies one from another. It is important to note that the complexity of this environment is possible even with each agent in the simulation following the same model (though with different probabilities). If the models varied, the inference engines (both the EE and the SIE) would be even more effective because they would have more diversity in the observations provided for inference.

7.3.2 Strategic Simulation

The simulation starts by reading in the data files that describe the target environment. Recall that the simulation engine is a multi-agent multi-team simulator, so even though the particular application is 5-vs-5 speedball paintball, it could easily be converted to run any similar situation with any number of teams, players, policies, or strategies. The framework only needs the requisite data: multi-layered models, team rosters and breakdown, agent roles, and any strategies or policies provided by domain knowledge. It can then take this information and implement the entire simulation without any other reconfiguration of the files necessary.

For this environment the field was read in first. This file describes each position on the field, each obstacle, and everywhere an agent could stand or move (this is a graph that contains the Total Movement Dependency Diagram). It describes the other positions observable from each of these locations as well. The rosters are loaded next. This is one continuous file that lists each player with their requisite inherent traits (e.g., amount of ammunition, rate of fire, etc.). Once loaded, the players are then recruited onto teams (2 teams in this case, but any number is possible). With the teams configured and players assigned, the simulation reads in the list of available policies and strategies. Recall that the system does not require any pre-built policies or strategies to run, but it will require a default model to handle the agents. Additionally, the system will always hold one extra slot for both policies and strategies so that it can learn as it goes. Once this new policy or strategy is observed and learned, it is added to the files and available as a selection the next time the simulation is run. Now that the system is fully configured, the simulation assigns strategies to each team. These strategies then match players with player types (e.g., commander, sniper, etc.) and assigns an initial policy to each. Each player is then moved to their home position and simulation begins.

The simulation is running a field imitating the 5-vs-5 Professional Speedball Paintball tournament from Phoenix, 2009. The field, shown from the television broadcast in Figure 7.6, is seen in the diagram of Figure 7.7. The simulation places the agents initially in home base, just as in the real-world game. This is the end of the initialization process of the simulation: all teams have been assigned strategies,

all agents on a team have been assigned policies (and, thus, behaviors), and the field is created and the simulation ready to start. Figure 7.9 shows the progress of the simulation after one member from each team has been eliminated, Figure 7.10 shows the progress once there are 4 members on one team and 2 on the other, and Figure 7.11 shows even further progression. Each agent is moved to their next location along this 3D modeled field where they check for opposing agents, scan for observations, fire on opposing agents, and take defensive action if necessary. The simulation is built to run without a GUI for maximum flexibility, configurability, and power; however, this 3D visualizer allows for better observation, learning, and interaction with both technical and non-technical users of SiMAMT. The 3D model is built in Unity (which also runs the simulation, configures the environments, and reports the results). This visualizer will be expanded in future research to make it more robust, flexible, and affordable.

Figure 7.6: Exemplar Field of Play for 5v5 PSP

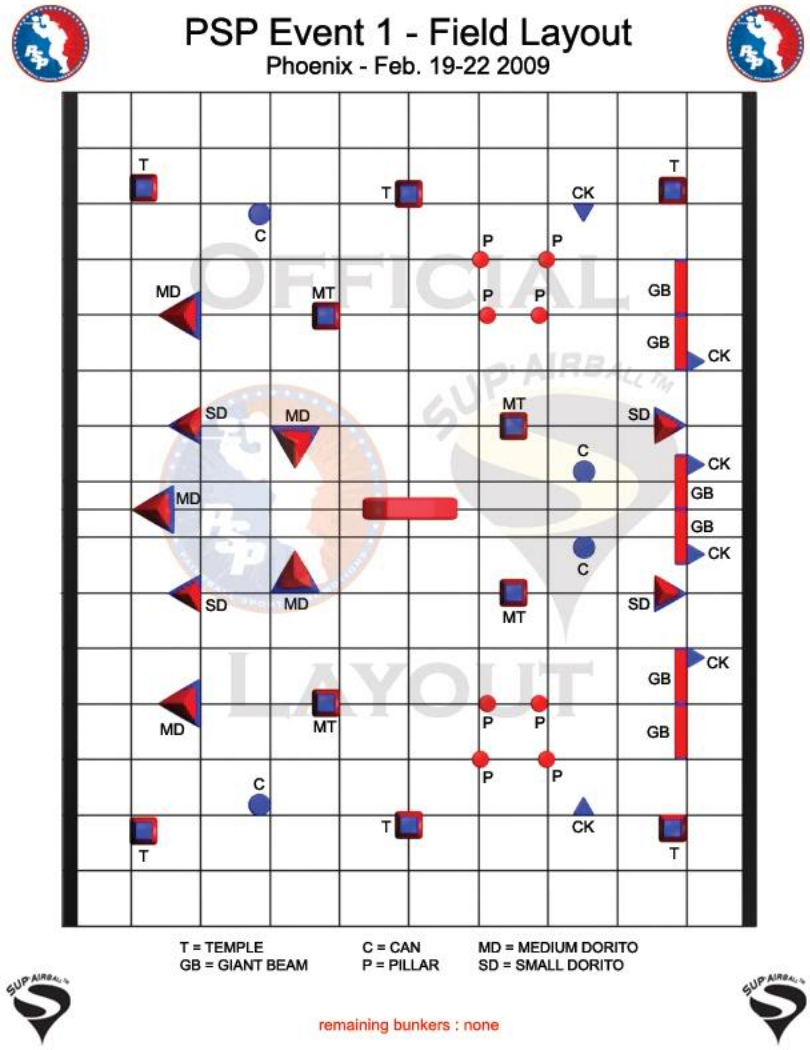


Figure 7.7: Field Diagram for 5v5 PSP

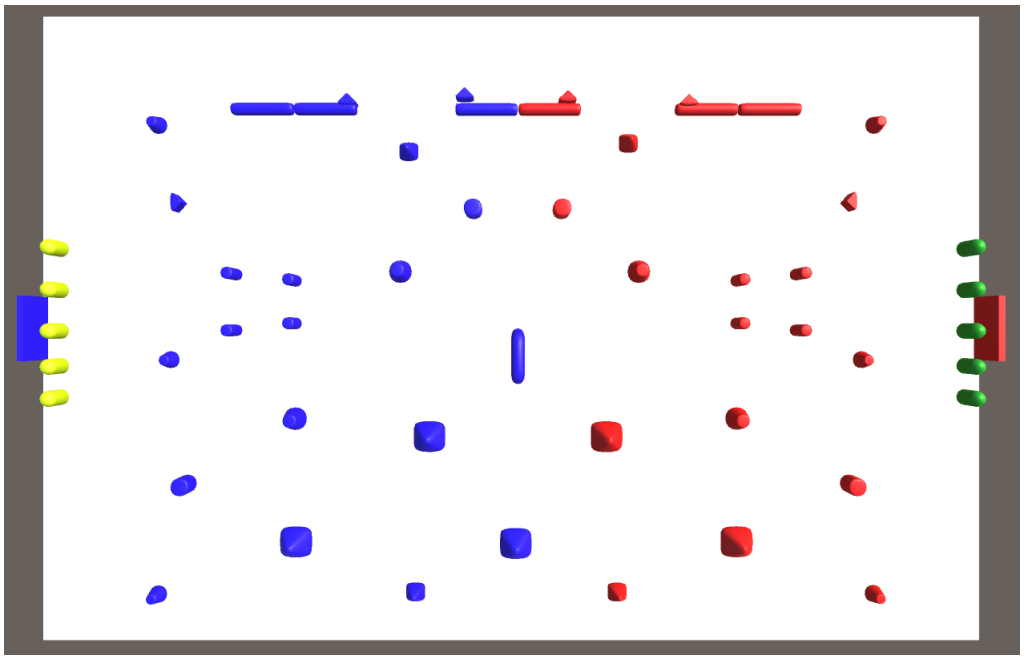


Figure 7.8: SiMAMT Simulation: 5v5 Starting Positions

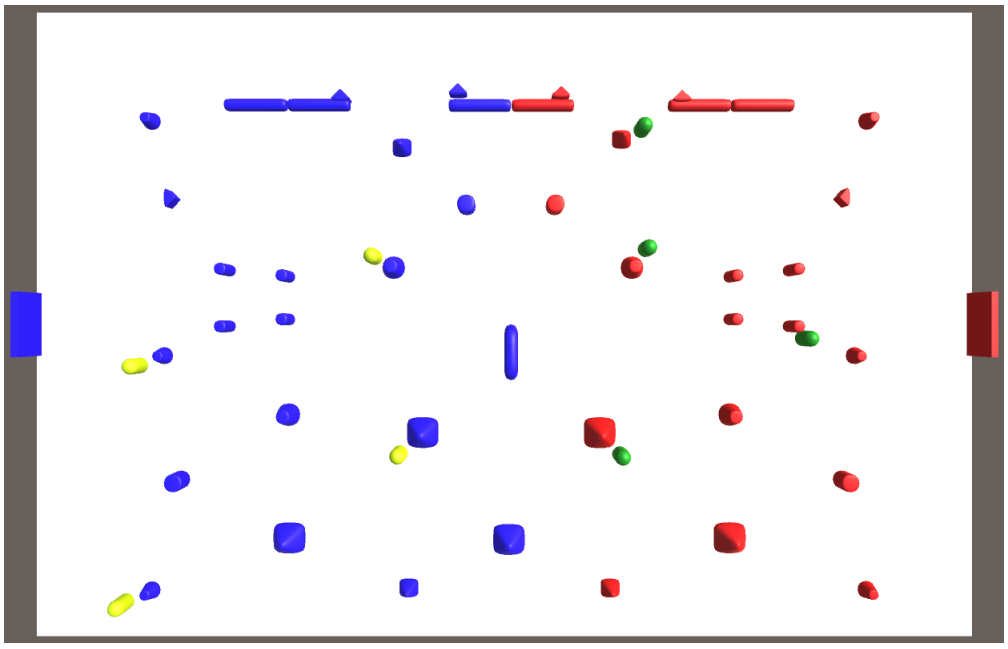


Figure 7.9: SiMAMT Simulation: 5v5 becomes 4v4

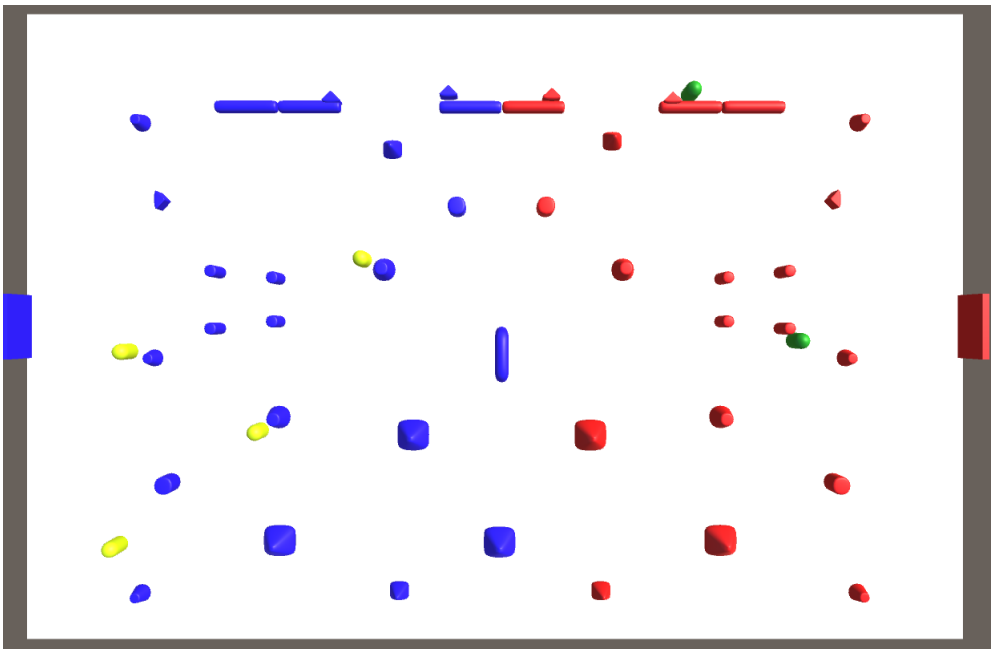


Figure 7.10: SiMAMT Simulation: 5v5 becomes 4v2

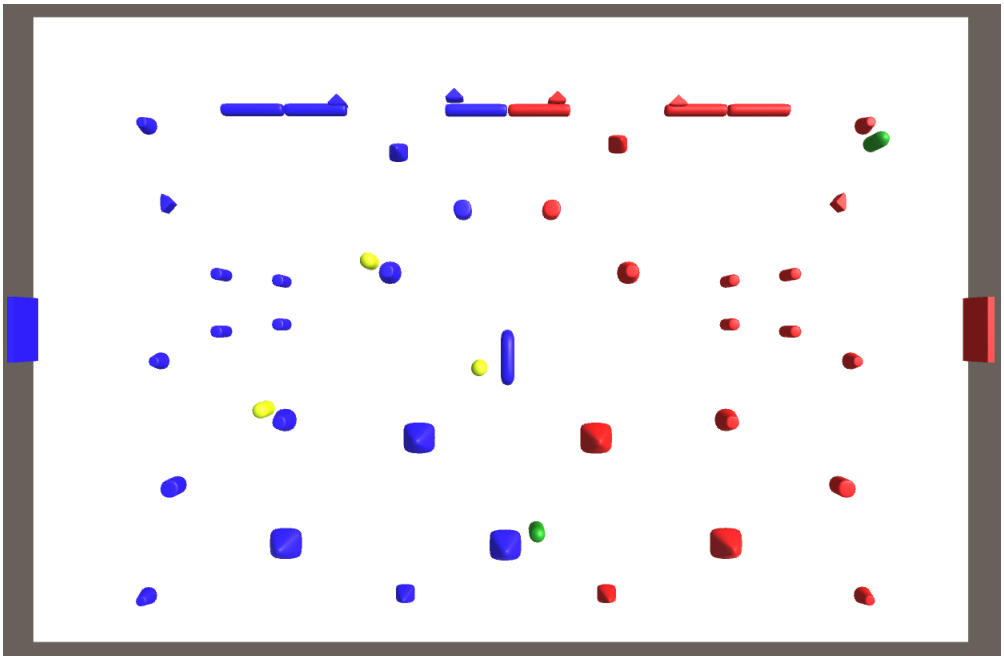


Figure 7.11: SiMAMT Simulation: 5v5 becomes 3v2

7.3.3 Evaluation Engine

The framework specification for the Evaluation Engine (EE) is used in this experiment, shown in Equation 7.11.

$$EE =< \sigma, \mathcal{A}, \mathcal{M}, \mathcal{D}, \Phi, \Psi, \Pi_s >, \Pi_s \subseteq \Pi \tag{7.11}$$

The EE is tasked with ensuring that the best performing policies are in place within a strategy. To do this, it compares the performance of each policy within the subset of policies available to the strategy to see which are the top performers. If there are any policies in force that are not performing as well as other policies within the strategy then they are candidates for replacement. There are several factors for this replacement, as mentioned previously, like the ϵ factor (the threshold variable to determine if the difference between two policies is significant enough to make a change), and the team factorization (the roles assigned to the team may override the need to to switch policies, or such a switch may actually hinder performance because of a well-known issue with a certain player utilizing that policy). If these factors do not dissuade the EE from its intention, a switch of policy is implemented. When the EE switches policies the behavior of the agent instantly changes because they are now going to be assigned a new behavior by the policy. When this occurs, the movement diagrams for each of the behaviors must be compared to see how to physically move the agent to the new movement diagram. If there is a corresponding junction then

the agent is cycled through their current movement diagram to get to the junction. If not, a closest point (using any of the many standard distance techniques) is located and a temporary move is initiated between the two points closest to each other on the two movement diagrams. If this is not practical, or if different supplies are needed for example, then the movement diagrams can be centered back at home base so that the agent cycles back there, restocks or resupplies, and then moves along the new movement diagram. In any of these cases the behavior does not officially switch until control can be handed off.

By way of example, consider a forward in soccer moving to the goalie position. This change in role (here, a change in policy) creates an entirely different behavior (playing forward vs. playing goalie). As a result, the framework can send both players to the bench, switch them in the field, or have them seamlessly switch the next time they are close to each other. This behavior is directed by the initial models employed in the system and the framework can work using any of these methods (or could be modified to use any other reasonable method).

7.3.4 Expansion of the Strategy Inference Engine

The validity and performance of the SIE has already been proven in earlier experiments, though in a simpler system. The main thrust of this research is to grow the inference engine to perform in the much more complex environment of SiMAMT. This simulation provides much less favorable conditions as noted above. The experiments pit the same 16 players, placed on the same two teams (Red and

Blue), against one another while varying the strategies that each team is using. A match is configured as two teams, 8 player max per team, 5 active players per team, and 1 - 3 hits to eliminate a player from the contest. As a result, even though the players do not change, they are assigned differing roles, differing overlying strategic factorizations, differing policies to follow, and each match is still probabilistically projected. Figure 7.12 shows a sample of three of the nine available strategies with a sample of the available policies (there are 26 policies in this experiment) that belong to each strategy. The strategy shows player types and the policies from which they select. The results of the various strategies competing with each other can be seen in Table 7.1. This chart shows the various wins and losses along with a percentage for comparison. There is a nice variety to the results which confirms the diversity and relative equality of merit for each strategy. Table 7.2 shows that each strategy has at least one strategy which it bested, and several have more than one (compared head-to-head). It is interesting to note that the number of match repetitions changes the landscape of this result dramatically. The results shown are the stabilized results after 100 repeated matches between each strategy (180 evaluations each, 360 head-to-head comparisons minus ties).

7.3.5 Intelligent Strategy Selection Engine

The framework specification for the Intelligent Strategy Selection Engine (ISSE) is used in this experiment, shown previously in Equation 7.10.

Strategy	Win	Loss	Win%	Loss%
0	71	109	39.4	60.5
1	72	108	40.0	60.0
2	86	94	47.8	52.2
3	104	76	57.8	42.2
4	127	53	70.6	29.4
5	110	70	61.1	38.9
6	76	104	42.2	57.8
7	78	102	43.3	56.7
8	108	72	60.0	40.0

Table 7.1: Strategy Evaluations (n=100)

Strategy	PlayerType	Policies		
StrongDefense	Heavy	BackDefenseC	BackDefenseR	
	Heavy	MidDefenseL	BackDefenseR	
	Commander	MidDefenseR	MidDefenseC	FrontDefenseL
	Rifleman	FrontDefenseC	FrontDefenseR	
	Rifleman	MidAttackL	FrontDefenseR	
StrongOffense	Rifleman	MidAttackC	MidAttackR	
	Rifleman	FrontAttackL	MidAttackR	
	CloseCombat	FrontAttackCL	FrontAttackCR	
	CloseCombat	Snake	FrontAttackR	
	Sniper	RightCrossLeft		
Balanced	Heavy	BackDefenseC	BackDefenseR	
	Heavy	MidDefenseL	BackDefenseR	
	CloseCombat	MidAttackC	MidAttackR	
	CloseCombat	FrontAttackL	MidAttackR	
	Sniper	RightCrossLeft		

Figure 7.12: Strategy Samples (3 of 9)

By examining the output of the EE and the SIE (their evaluations of the policies in place and the strategies in place, respectively), the ISSE is monitoring the strategies in place. By comparing the performance of the strategy that the team is currently using with all available strategies a decision can be made on whether to not to switch to a better performing strategy. Additionally, with the feedback from the tree-matching of the SIE, the ISSE can determine that, based on the performance of the most likely opponent strategy, a strategy change is necessary. Again, as with the EE, a threshold must be met to make the change occur. Either of these two methods can initiate a team strategy change in an automated fashion based on the relative

Strat	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**0**	**27**	**27**	**65**	**35**	**29**	**81**	**7**	**23**	**42**
1	-27	**0**	-23	-20	-52	-36	**17**	-27	-71	-43
2	-27	**23**	**0**	**7**	-19	**6**	**50**	-9	-34	**14**
3	-65	**20**	-7	**0**	-1	**8**	**46**	-9	-12	-10
4	-35	**52**	**19**	**1**	**0**	-13	**34**	-24	-46	-13
5	-29	**36**	-6	-8	**13**	**0**	**45**	-6	-27	-14
6	-81	-17	-50	-46	-34	-45	**0**	-44	-81	-48
7	-7	**27**	**9**	**9**	**24**	**6**	**44**	**0**	**7**	**14**
8	-23	**71**	**34**	**12**	**46**	**27**	**81**	-7	**0**	**32**
9	-42	**43**	-14	**10**	**13**	**14**	**48**	-14	-32	**0**

Table 7.2: Strategy Evaluations (n=100)

performance indicated; however, there is also a third case to consider. If there is provided to the system any domain knowledge or prior experience to know which of the strategies available may perform best against the most likely opponent strategy then this can also trigger a change of strategy. In this case, once the SIE has given a clear indication of the most likely strategy being used by another team (or teams), the ISSE can compare this indication with its list of strategies and counter-strategies. If there is a known counter-strategy then that will be offered as the candidate for the next strategy to be implemented by the team.

The change of strategy is more complicated than a change of policy as it affects every member of the team. First, there is a comparison done to see if any roles in the new strategy match any of the roles on the old strategy. If this is the case, these assignments remain unchanged. Second, each agent is observed for critical function

to make sure that switching their policy (the result of switching strategies is that each team member is assigned a new policy) will not cause undue harm (losing the game, violating a rule, etc.). If so, the agents are cycled through the change so that another agent can take over this critical function and the simulation can continue. Third, if not, then each player is assigned a new policy by the new strategy. Each agent then changes its behavior, as prescribed by the policy, as soon as it is able (as documented above in the EE).

By way of example, consider that a basketball team is currently running a strategy called *HeavyOffense*. As a result, they are winning the game. However, as the game nears its conclusion, the other team initiates a strategy called *FoulToSaveClock*. This strategy has their team members foul the other players in an attempt to conserve clock time. As a result, the first team may realize that a change to *HighFT* (a strategy that puts the best free throw players out on the court) or a change to *HandsTeam* (a strategy that puts in the best ball-handlers) is best. If so, a change is initiated by calling a time out in the game and resetting the player roles (including roster changes).

7.4 Complexity / Similarity Analysis

Before showing the strategy inference results for this strongly complex environment, it is imperative to understand the overlapping nature of the policies and the strategies. Each position appears on average 4.10 times ($\sigma = 5.4$) in each policy, and if we exclude home base, which is part of every policy, the average is 2.88 ($\sigma = 1.9$). Additionally,

each policy appears an average of 2.96 times ($\sigma = 1.7$) in the list of strategies, so there is a lot of variance and spread. These results help to understand just how difficult it is to recognize any one policy, much less a particular strategy, when there is so much similarity. Nevertheless, the goal was to show the SIE in a realistic environment rather than one particularly configured to its advantage. In fact, this particular setup is a kind of 'torture test' for the SIE because of so much similarity, so little data due to the length of the rounds, and the difficulty of observation in an obstacle-rich dynamic environment.

7.5 Strategy Inference Experimental Results

Even with this complexity and similarity, Table 7.3 shows the effectiveness of the inference and the increased awareness and success of utilizing it. It shows the correct strategic inference for multiple stochastically generated simulations. Given 81 different strategic interactions, the second column shows the number recognized correctly (with accuracy in column 4). The last column shows the number of steps into the simulation before the correct strategy was recognized. This shows a consistent and measurable increase over the single-factor strategy inference done earlier. These experiments prove that the SIE can, even in environments where there is significant overlap, correctly determine the strategy that an opposing team is using and can do so in interactive time (e.g., in an average of 25 steps, less than 0.01 seconds on even a basic computer).

# of Strat	Correct	Total	Accuracy	Steps to Recognize
10	72	81	88.89%	22
10	77	81	95.06%	27
10	74	81	91.35%	24

Table 7.3: Strategy Inference

The experiments show that the overall framework is a success. First, the strategic representation selected and implemented was effective. The experiments show that the model-based representation schema for strategic interaction can hold the information needed to govern the actions of the agents. Further, the hierarchical nature of the modeling proved effective at maintaining separated operations per level while affecting proper interactions within each level. Second, the simulation shows veracity and efficacy. The agents within the simulation accurately model the strategic behavior intended and work cohesively as a team. The team's efficacy increases when they are operating as a unit. The simulated games play out very similarly to exemplar professional matches, though the complexity of the real-world prohibits one-to-one correlation. However, the teams that are following proven strategies in the real-world and in the simulation succeed in similar ratios. The simulation does show strong benefit from strategic interaction, cohesive unit behaviors, and improved success when following superior strategies to those in play. This leads to the third element, strategic inference. The strategic inference is effective on multiple fronts: it consistently recognizes strategies being used by other teams, it recognizes them in interactive time, and it can switch strategies for increased performance and success.

Overall, the framework is able to accomplish all of this even in complex multi-agent multi-team systems.

7.6 Conclusions

This ability to rapidly and accurately determine which strategy the opponent is following allowed the previously underperforming strategies (those that lost significantly against certain strategies) to reverse the outcome of the simulations. The ISSE would switch the strategy to one that better matches that opponent's strategy resulting in a dramatic increase in wins for that strategy. To examine this in context, a poor strategic matchup was created, like assigning Team A to strategy 1 and Team B to strategy 3. When left alone, Team A would lose, on average across the 100 matches, by 23 (recall that the probabilistic nature of the simulation allows for variance in outcome even when using the same strategies over and over). Because Team A can recognize that Team B is using strategy 3 (having been informed by the SIE), the ISSE then switches Team A to strategy 0. Team A will now switch from a -23 differential in wins and losses to a +65 differential, a remarkable improvement. At this point, even if Team B switches itself to another strategy, Team A will also keep switching to maintain its advantage. *This final element is strong confirmation of the total contribution of the SiMAMT framework - it models strategy with high-fidelity, it recognizes strategies that are being used in interactive time, and it switches to gain an advantage over other teams in this multi-agent multi-team environment.*

Chapter 8

Conclusions and Future Work

8.1 Conclusions

The data show that the simulation is a success. The strategies are able to play through the simulation and compete with an intense amount of expressivity and complexity. The interaction of the agents is faithful to the policies and the strategies. The strategies show diversity and performance appropriate to their configuration. The configuration is flexible to allow for a wide variety of strategic situations with only a reconfiguration of the data files (no code changes). The simulation also provides a wealth of data to understand the performance of the various agents, teams, policies, and strategies. This analysis shows the power of the simulation and its fidelity to the modeled environment. Finally, the strategic inference works, even within the confines of such a complex and indistinguishable system. The key take away is the ability for a team to utilize this strategic inference to quickly adjust their own strategy to maintain maximum advantage. This recognition is happening in interactive time without slowing the simulation down and the decision making is reliable.

8.2 Future Work

The framework has proven itself a success. The representations have proven faithful to the real-world concepts they are attempting to embody, the strategic reasoning is working at multiple levels, and the systems encoded within SiMAMT can both reason and learn. Each experiment, those in part that proved the individual elements, and those in totem, that prove the efficacy of the framework, have delivered results that make it clear that the systems are working and the goals have been accomplished.

In the future, it would be great to expand the system to an even greater scale through raising the numbers of the levels within the hierarchy. This could be utilized to model several well-known battles from history, like the aforementioned Battle of Waterloo, and examine first the veracity of the system to model such complex interaction; second, to experiment with alternatives to see if several well-known historical critiques would have produced different outcomes; third, to expand the scenario to include additional data like environmental conditions. Another area to expand would to experiment with different communication models within the teams to see how they work and how they affect tactics and results. Finally, the work could be expanded to model many different kinds of systems to prove the flexibility of the framework. This work is already being co-opted for use in the Energy Field by the Department of Energy and for small-group tactics analysis by The Army Research Laboratory. Each of these uses is proof-positive of the veracity, efficacy, and applicability of the framework to solve real-world problems.

8.3 Additional Experiment: Machine Learning in Stochastic Environments

This experiment seeks to examine learning in an environment where the variables, methodologies, and rewards can vary. The normal learning techniques will either fail to converge in such environments or will take an inordinate amount of time to converge. The experiment uses two agents, one under computer control and the other under the control of an artificial intelligence, that are attempting to maximize their reward. There are three control scenarios: first, the agents try to converge; second, they try maintain an equidistant position; third, they try to avoid each other. The reward in each scenario is based on accomplishing the goals listed. The challenge is that the scenario in play is randomized, and thus the AI agent perceives itself to be in a non-deterministic environment where the same state and action choices result in different rewards. The experiment attempts to learn in this environment using standard machine learning techniques and comparing their performance. Then the AI agent will shift into a strategy-aware mode where it is taught the three strategies and then attempts to determine the strategy of the other agent and mimic it to maximize reward. This will test the hypothesis that strategic reasoning allows an AI to perform optimally in an environment where standard machine learning techniques either perform poorly or fail to converge. One additional consideration is that the machine learning techniques will need to have sufficient training time before their performance is evaluated to make up for the strategic advantage. This advantage may be viewed by the machine learning techniques as a form of bootstrap learning

and would give the strategic method a perceived advantage. The reality of this advantage will also be tested and reported. These experiments are underway.

Bibliography

Bibliography

AllExperts (2016). AllExperts Answers Speedball Position Inquiry.

ARToolkit, I. (2008). ARToolkit, University of Washington, HITLab, http://www.hitl.washington.edu/artoolkit.

Associated Press (2012). U.S. Soldiers take an overwatch position.

Bowling, M. and Veloso, M. (2001). Rational and convergent learning in stochastic games. In *International Joint Conference on Artificial Intelligence*, volume 17, pages 1021–1026. LAWRENCE ERLBAUM ASSOCIATES LTD.

Bowling, M., Veloso, M., et al. (2004). Existence of multiagent equilibria with limited agents. *J. Artif. Intell. Res. (JAIR)*, 22:353–384.

BPAA (2017). Play diagrams.

Chang Wook Kim and Seongwon Cho and Choong Woong Lee (1995). Fast competitive learning with classified learning rates for vector quantization. *Signal Processing" Image Communications*, 6:499–505.

Chernova, S. and Veloso, M. (2009). Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 34(1):1.

Chung, F. L. and Lee, T. (1994). Fuzzy competitive learning. *Neural Networks*, 7(3):539–551.

Das, B., Datta, S., and Nimbhorkar, P. (2013). Log-space algorithms for paths and matchings in k-trees. *Theory of Computing Systems*, 53(4):669 – 689.

Datta, S., Kulkarni, R., and Roy, S. (2010). Deterministically isolating a perfect matching in bipartite planar graphs. *Theory of Computing Systems*, 47(3):737 – 757.

Fernández, F., García, J., and Veloso, M. (2010). Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems*, 58(7):866–871.

Fernández, F. and Veloso, M. (2005). Probabilistic policy reuse. *Journal of Artificial Intelligence Research*.

Fernández, F. and Veloso, M. (2006). Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 720–727. ACM.

Fukuda, K. and Matsui, T. (1994). Finding all the perfect matchings in bipartite graphs. *Applied Mathematics Letters*, 7(1):15 – 18.

Greenwald, A., Hall, K., and Serrano, R. (2003). Correlated q-learning. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, volume 20, page 242.

Han, K., Veloso, M., et al. (2000). Automated robot behavior recognition. In *ROBOTICS RESEARCH-INTERNATIONAL SYMPOSIUM-*, volume 9, pages 249–256.

Headquarters, D. o. t. A. (2013). Fm 3-0 operations. *GPO Publications*.

Hu, J., Wellman, M., et al. (1998). Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning*, volume 242, page 250. Citeseer.

IPRE (2007). Institute for Personal Robots in Education, www.roboteducation.org.

J. R. Hoare and R. Edwards and B. MacLennan and L. Parker (2011). Myro-C++: An Open Source C++ Library for CS Education Using AI. *Proceedings of the 24th International Conference of the Florida Artificial Intelligence Research Society (FLAIRS 2011)*.

Jorg Bewersdorff and Translated by David Kramer (2005). *Luck, Logic, and White Lies: The Mathematics of Games*. A. K. Peters, Ltd., 888 Worcester Street, Suite 230 Wellesley, MA 02482.

Kumar, R., Talton, J. O., Ahmad, S., Roughgarden, T., and Klemmer, S. R. (2011). Flexible Tree Matching. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three*, IJCAI'11, pages 2674–2679. AAAI Press.

Laviers, K., Sukthankar, G., Aha, D. W., Molineaux, M., Darken, C., et al. (2009). Improving offensive performance through opponent modeling. In *AIIDE*.

Littman, M. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, volume 157, page 163.

Matarić, M. (1996). Learning in multi-robot systems. *Adaption and Learning in Multi-Agent Systems*, pages 152–163.

Matarić, M. (1997). Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4(1):73–83.

Michael Bowling and Manuela Veloso (2002). Multiagent Learning using a Variable Learning Rate. *Artificial Intelligence*, 136:212–250.

Mohri, M., Pereira, F., and Riley, M. (2000). The design principles of a weighted finite-state transducer library. *Theoretical Computer Science*, 231(1):17 – 32.

Montgomery, B. (1968). Field-marshal viscount montgomery of alamein, a history of warfare. page Introduction.

Moore, E. F. (1956). Gedanken-experiments on sequential machines. *Automata studies*, 34:129–153.

Moore, F. R. (1971). On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. *IEEE Transactions on Computers*, C-20(10):1211–1214.

MU, W. (2012). Technological information, http://wordpress.morningside.edu/cdl001/rpsls/.

Muhlestein, D. (2014). How to Move: Aggressive Movement is the Key to Dominating at Paintball.

Office, N. S. (2015). Nato glossary of terms and definitions, aap-06 edition 2015. *NATO SDO Publications*.

Oxford University Press. OED Online.

Oxford University Press. Oxford Dictionaries.

Parallax, I. (2012). Scribbler Robot, from Parallax, http://www.parallax.com/ScribblerFamily/tabid/825/Default.aspx.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference (2nd ed.)*. Morgan Kaufmann, San Francisco, CA.

Publications, J. D. (2008). Joint doctrine publication 0-01 british defence doctrine, 3rd edition. *Joint Doctrine Publications*.

Rabin, M. O. and Scott, D. (1959). Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125.

Russell, S. and Norvig, P. (2003). Artificial intelligence: A modern approach. pages Chp. 12,17.

Sepp, S. (2015). Homeomorphic graph images.

Simon Parson and Michael Woolridge (2002). Game Theory and Decision Theory in Multi-Agent Systems. *Autonomous Agents and Multi-Agent Systems*, 5:243–254.

Soumya Paul and R. Ramanujam (2010). Imitation in Large Games. *Proceedings of GandALF 2010, EPTCS 25*, pages 162–172.

Soumya Paul and R. Ramanujam (2011a). Dynamic Restriction of Choices: Synthesis of Societal Rules. *LORI 2011, LNAI 6953*, pages 28–50.

Soumya Paul and R. Ramanujam (2011b). Neighbourhood Structure in Large Games. *TARK 2011, ACM, ISBN 978-1-4503-0707-9*.

SpecialOpsPaintball (2012). Paintball Team Positions.

Sports, C. (2013). Strategies and Tactics for Paintball.

Staff, X. O. (2015). Pass routes 101.

Stone, P. and Veloso, M. (1999). Task decomposition and dynamic role assignment for real-time strategic teamwork. *Intelligent Agents V: Agents Theories, Architectures, and Languages*, pages 293–308.

Tutorials, F. (2015). Ultimate football plays.

University, U. A. F. A. (2012). School of advanced air and space studies: Saas history. *US Air Force Publications*.

Vazirani, V. V. (1989). {NC} algorithms for computing the number of perfect matchings in k3,3-free graphs and related problems. *Information and Computation*, 80(2):152 – 164.

Warpig.com (2016). PSP Field Layout Pheonix 2009.

Weber, B. G. and Mateas, M. (2009). A data mining approach to strategy prediction. In *2009 IEEE Symposium on Computational Intelligence and Games*, pages 140–147.

Weiss, G. and Sen, S. (1995). *Proceedings of the Workshop on Adaption and Learning in Multi-Agent Systems*. Springer-Verlag.

Wilson, Andrea (2002). Bounded Memory and Biases in Information Processing. *NAJ Economics*, 5(3).

Windsor, A. (2015). Boost library: Planar graphs.

Yang, E. and Gu, D. (2004). Multiagent reinforcement learning for multi-robot systems: A survey. *Department of Computer Science, Univeristy of Essex, Tech. Rep.*